

Санкт-Петербургский Политехнический Университет Петра Великого  
Институт компьютерных наук и технологий  
Кафедра компьютерных систем и программных технологий

# Отчет по лабораторным работам

Сети ЭВМ и телекоммуникации

Программирование сокетов протоколов TCP и UDP

**Работу**

**выполнила:**

Городничева Л.В.

Группа: 43501/3

**Преподаватель:**

Алексюк А.О.

Санкт-Петербург

2017

# 1. Цель работы

Изучение принципов программирование сокетов с использованием протоколов TCP и UDP.

# 2. Программа работы

TCP:

- Реализация простейшего TCP сервера и клиента на ОС Linux и Windows.
- Реализация многопоточного обслуживания клиентов на сервере.
- Реализация собственного протокола на основе TCP для индивидуального задания (сервер на Windows, клиент на Linux).

UDP:

- Модификация сервера и клиента для протокола UDP на ОС Windows и Linux.
- Реализация собственного протокола на основе UDP для индивидуального задания (сервер на Linux, клиент на Windows).
- Обеспечение надежности протокола UDP, посредством нумерации пакетов и посылки ответов

Дополнительное задание:

Исследовать реальные прикладные протоколы. Необходимо “притвориться” клиентом и подключиться к одному из существующих общедоступных серверов. Использовать утилиты nc, telnet, PuTTY или другие им подобные, чтобы подключиться к серверу. Нельзя использовать специализированные клиенты для конкретного протокола! Необходимо вручную отправлять команды серверу в соответствии с протоколом. Нужно выполнить следующие опыты:

- Подключиться к HTTP-серверу и запросить веб-страницу
- Подключиться к FTP-серверу, запросить список файлов в директории и получить файл
- Подключиться к SMTP-серверу и попробовать отправить письмо
- Подключиться к POP3-серверу, попробовать проверить почту и получить письмо

В отправляемом и принимаемом письме должно быть мое имя и фамилия В ходе опытов необходимо попробовать как незащищенное plaintext-подключение, так и шифрованное TLS-подключение.

### 3. Простейший TCP сервер и клиент

Для создания сокета в библиотеках BSD-socket и WinSock имеется системный вызов `socket`:

Листинг 1: Вызов `socket`

```
1 int socket(int domain, int type, int protocol);
```

Для установления TCP-соединения используется вызов `connect`:

Листинг 2: Вызов `connect`

```
2 int connect(int s, const struct sockaddr* serv_addr, int addr_len);
```

Результатом выполнения функции является установление TCP-соединения с TCP-сервером.

Передача и приём данных в рамках установленного TCP-соединения осуществляется вызовами `send` и `recv`:

Листинг 3: Вызов `send` и `recv`

```
3 int send(int s, const void *msg, size_t len, int flags);  
4 int recv(int s, void *msg, size_t len, int flags);
```

Параметр `s` – дескриптор сокета, параметр `msg` – указатель на буфер, содержащий данные (вызов `send`), или указатель на буфер, предназначенный для приёма данных (вызов `recv`). Параметр `len` – длина буфера в байтах, параметр `flags` – опции отправки или приёма данных.

Возвращаемое значение – число успешно посланных или принятых байтов, в случае ошибки функция возвращает значение -1.

Завершение установленного TCP-соединения осуществляется в библиотеке BSD-socket с помощью вызова `shutdown`:

Листинг 4: Завершение соединения

```
5 int shutdown(int s, int how);
```

По окончании работы следует закрыть сокет, для этого в библиотеке BSD-socket предусмотрен вызов `close`:

Листинг 5: Вызов `close`

```
6 int close(int s);
```

Структура TCP-сервера:

1. Создание сокета с помощью вызова `socket`. При этом локальные адрес и порт сокету назначаются из числа свободных;
2. Привязка сокета к удаленному адресу с помощью вызова `bind` (для сервера устанавливается адрес `INADDR_ANY`, который позволяет получать данные с любых адресов);
3. Перевод сокета в состояние прослушивания соединений с помощью вызова `listen`. При этом данные передавать через сокет нельзя, единственная его задача – получение запросов на соединение;
4. Прием соединений клиентов с помощью вызова `accept`. Данный вызов блокирует выполнение потока, пока не придет соединение от клиента, в результате которого создается новый сокет, связанный с адресом клиента;
5. Далее происходит прием/передача данных с помощью `recv/send`.
6. Закрытие сокета с помощью `shutdown` и `close`

Структура TCP-клиента:

1. Создание сокета с помощью вызова `socket`. При этом локальные адрес и порт сокету назначаются из числа свободных;
2. Установка соединения с помощью `connect`
3. Далее происходит прием/передача данных с помощью `recv/send`.
4. Закрытие сокета с помощью `shutdown` и `close`

Простейшее клиент-серверное приложение, реализующее описанные структуры и использующее стандартные функции управления сокетами, приведены в приложении (лис.7 и лис.8).

В Windows вместо вызовов `write` и `read` происходят вызовы `send` и `recv`.

## 4. Простейший UDP сервер и клиент

В случае установленного адреса по умолчанию для протокола UDP (вызов `connect`) функции для передачи и приёма данных по протоколу UDP можно использовать вызовы `send` и `recv`. Если адрес и порт по умолчанию для протокола UDP не установлен, то параметры удалённой стороны необходимо указывать или получать при каждом вызове операций записи или чтения. Для протокола UDP имеется два аналогичных вызова `sendto` и `recvfrom`:

### Листинг 6: Передача и прием данных

```
7 int sendto(int s, const void *buf, size_t len, int flags, struct sockaddr *to,  
    ↪ int* tolen);  
8 int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, int  
    ↪ * fromlen);
```

Структура UDP-сервера:

1. Создание сокета с помощью вызова `socket`. При этом локальные адрес и порт сокету назначаются из числа свободных;
2. Привязка сокета к удаленному адресу с помощью вызова `bind` (для сервера устанавливается адрес `INADDR_ANY`, который позволяет получать данные с любых адресов);
3. Далее происходит прием/передача данных с помощью `recvfrom/sendto`.
4. Закрытие сокета с помощью `close`

Структура UDP-клиента:

Структура UDP-клиента ещё более простая, чем у TCP-клиента, так как нет необходимости создавать и разрывать соединение.

1. Создание сокета с помощью вызова `socket`. При этом локальные адрес и порт сокету назначаются из числа свободных;
2. Привязка сокета к удаленному адресу с помощью вызова `bind`
3. Установка соединения с помощью `connect`
4. Далее происходит прием/передача данных с помощью `recv/send`.
5. Закрытие сокета с помощью `shutdown` и `close`

Простейшее клиент-серверное приложение, реализующее описанные структуры и использующее стандартные функции управления сокетами, приведено в приложении (лис.9 и лис.10).

## 5. Многопоточное обслуживание клиентов на сервере

Для организации работы сервера с несколькими клиентами необходимо создавать новый сокет для каждого из клиентов, а старый созданный сокет сделать слушающим: сокет будет использоваться только в `listen` и `accept` для подключения новых клиентов.

Подключение клиентов необходимо сделать в цикле. После подключения каждого нового клиента выделять отдельный поток для общения с ним. В этом потоке должна вызываться функция работы с клиентом.

Многопоточный ТСП-сервер приведен в приложении (лис.11)

В приведенной выше программе видно вызов асерт в бесконечном цикле, а также создание нового потока, который начинает работу с функции `formulti`, передавая ей `newsockfd`.

## **6. Индивидуальное задание**

### **6.1. Задание**

Разработать клиент-серверную систему дистанционного тестирования знаний, состоящую из централизованного сервера тестирования и клиентов тестирования.

### **6.2. Основные возможности**

Серверное приложение должно реализовывать следующие функции:

1. Прослушивание определенного порта
2. Обработка запросов на подключение по этому порту от клиентов
3. Поддержка одновременной работы нескольких клиентов через механизм нитей
4. Регистрация клиента, выдача клиенту результата его последнего теста, выдача клиенту списка тестов
5. Получение от клиента номера теста
6. Последовательная выдача клиенту вопросов теста и получение ответов на вопросы
7. После прохождения теста – выдача клиенту его результата
8. Обработка запроса на отключение клиента
9. Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1. Установление соединения с сервером
2. Посылка регистрационных данных клиента

3. Выбор теста
4. Последовательная выдача ответов на вопросы сервера
5. Индикация результатов теста
6. Разрыв соединения
7. Обработка ситуации отключения клиента сервером

### 6.3. Настройки приложений

Разработанное клиентское приложение должно предоставлять пользователю возможность введения идентификационной информации, настройки IP-адреса или доменного имени, а также номера порта сервера информационной системы.

Разработанное серверное приложение должно предоставлять пользователю возможность настройки начальной точки входа в информационную систему каждого пользователя.

### 6.4. Прикладной протокол ТСР

После подключения клиента к серверу, сервер ждет команд от клиента. Существует 5 возможных команд от клиента.

Команда имеет вид:

<команда> <аргумент>

Для всех отправляемых сообщений установлена фиксированная длина в 20 байт. Если необходимо послать больше, то отсылается несколько пакетов по 20 байт.

Поддерживаемые команды клиента:

1. end - завершение работы клиента
2. register - регистрация клиента (выполняется в две команды, после нажатия enter клиенту необходимо ввести <логин> <пароль>)
3. show - выводение списка существующих тестов
4. getResult - выводение результата последнего пройденного теста
5. getTest <номер теста> - выводение списка вопросов определенного теста

Если клиентом введена несуществующая команда, то ничего не происходит.

Ответы сервера на команды клиента:

1. Нет сообщения от сервера
2. "You are successfully logged in! / Try another password or login! / You are successfully registered / Wrong number of argumenets. You can't create login and password with spaces"
3. Выводится список всех тестов, например "1 - Math test 2 - Phsycological test"
4. "Your last result was: [x out of y]"(где x - количество правильных ответов, y - количество всех ответов)
5. Выведение списка вопросов

Действия сервера:

1. Сервер закрывает сокет
2. — / — / Регистрация клиента (запись логина и пароля в файл registered.txt)
3. Посылка списка вопросов из файла list\_tests.txt
4. Посылка результата последнего теста клиента из файла registered.txt
5. Последовательная посылка списка вопросов из конкретного файла test<номер теста>.txt, а затем сверка ответов с правильными ответами теста из файла anstest<номер теста>.txt

Сообщения об ошибках:

1. recv failed with error: Ошибка при приеме данных
2. socket failed with error: Не удалось создать сокет
3. getaddrinfo failed with error: Не удалось получить адрес
4. ERROR reading from socket: Ошибка при приеме данных
5. accept failed with error: Ошибка при подтверждении подключения нового клиента
6. WSASStartup failed with error: Не удалось инициализировать структуру WSASStartup
7. bind failed with error: Ошибка привязки сокета к IP адресу
8. listen failed with error: Ошибка прослушивания сокета при ожидании новых клиентов
9. send failed with error: Ошибка при передаче данных



**Формат содержимого файла registered.txt (логины, пароли и последние результаты тестирования клиентов):**

<логин> <пароль> <x out of y>

**Формат содержимого файла list\_tests.txt (список тестов):**

<номер теста> - <имя теста>

**Формат содержимого файла test<номер теста>.txt (вопросы теста):**

QUESTION:

<вопрос>

ANSWERS:

<номер ответа> ) <ответ>

<номер ответа> ) <ответ>

<номер ответа> ) <ответ>

**Формат содержимого файла anstest<номер теста>.txt(правильные ответы на определенный тест):**

<ответ>

<ответ>

Поддерживаемые команды сервера:

1. end - завершение работы сервера
2. close - отключение определенного клиента
3. send - посылка сообщения клиенту
4. show - список подсоединившихся клиентов

## 6.5. Прикладной протокол UDP

В отличие от TCP, в прикладном протоколе UDP сервер поддерживает лишь одну команду: end - завершение работы сервера. В остальных командах отсутствует необходимость, т.к. в UDP не устанавливается соединение и все обмены со всеми клиентами происходят через один единственный сокет.

В начале каждого пакета записан его номер в текущем обмене. Далее через пробел следует само сообщение.

Сообщения об ошибках:

1. recv failed with error: Ошибка при приеме данных
2. socket failed with error: Не удалось создать сокет
3. getaddrinfo failed with error: Не удалось получить адрес

4. WSAStartup failed with error: Не удалось инициализировать структуры WSAStartup
5. bind failed with error: Ошибка привязки сокета к IP адресу

## 6.6. Описание архитектуры и особенности реализации TCP

Изначально происходит создание серверного сокета, который затем будет прослушивать подключения на определенном, заданном при создании, порту. Для того, чтобы сервер мог параллельно с установкой соединений самостоятельно писать и выполнять команды, создается новый поток (`acceptThreadFunction`), в котором как раз и осуществляется ожидание новых подключений. В основном потоке сервер принимает команды из командной строки, а затем осуществляет выполнение одной из следующих команд: завершение сервера, посылка сообщения определенному клиенту, закрытие определенного клиента и просмотр списка клиентов, подключенных в данный момент. Все операции с коллекцией (`poolOfSockets`), хранящей в настоящее время подключенные сокета, защищены мьютексами, так как к коллекции возможно одновременное обращение из нескольких потоков. В поточной функции `acceptThreadFunction` происходит ожидание новых подключений в цикле. Когда клиент подключается к серверу, клиентский сокет добавляется в коллекцию с активными сокетами (`poolOfSockets`), а на работу клиента выделяется отдельный поток (`workingFlow`), в который передается сокет для общения с клиентом. Поточная функция `workingFlow` ожидает команды от клиента, приведенные выше. В случае, если пришла строка об отключении клиента, клиентский сокет удаляется из коллекции сокетов и закрывается. Если же пришла строка о регистрации, выполняется функция регистрации, в которой изначально считываются строки из файла, чтобы была информация о существующих клиентах. Затем сервер ждет логин и пароль от клиента и сверяет их с уже существующими данными. Если такого логина, с которым пытается подключиться клиент, не существует, создается новая запись в файле, а клиенту посылается сообщение о том, что он зарегистрирован. Если же логин, с которым клиент пытается зарегистрироваться, уже существует в регистрационном файле, то существует 2 варианта последующих действий. Если пароль совпадает с тем, что имеется в регистрационном файле, то клиент считается залогиненным и его сокет заносится в коллекцию залогиненных сокетов, если же пароль не совпал, то либо клиент ошибся в пароле, пытаясь залогиниться, либо клиент при регистрации указал уже существующий логин. В последнем варианте клиенту посылается сообщение о том, что он ему стоит попробовать другой логин либо поробовать другой пароль. Считывание пароля с логином продолжается, пока клиент либо не зарегистрируется, либо не залогинится. Если пришла команда с запросом тестов, то название тестов считывается из определенного файла в одну строку. Может случится такая ситуация, что размер такой строки превысит длину единичной посылки, тогда такое сообщение разобьется на несколько посылок равно определенной протоколом длины и передастся клиенту. Если сервер получил команду на запрос результата клиента, то мы изначально

проверяем залогинен ли данный клиент, если он залогинен, то результат считывается из специального файла по логину и паролю клиента. Если пришла команда о выдаче определенного теста, также происходит проверка, залогинен ли пользователь, затем по номеру теста выбирается определенный файл с вопросами и вариантами ответов. Вопрос может превысить длину сообщения, заданную протоколом, поэтому сообщение разбивается на несколько сообщений длиной определенной протоколом. После отправки вопроса и вариантов ответа, сервер ожидает ответ клиента. Приняв ответ, записывает ответ в вектор и посылает следующую порцию вопрос-варианты ответов. Отправив все вопросы, сервер считывает верные варианты ответов из специального файла и сверяет с ответами клиента, формируя тем самым результат клиента. Этот результат записывается в файл с логином и паролем и посылается клиенту.

Когда сервер хочет завершить свою работу, ему необходимо отключить свой слушающий сокет, все клиентские сокеты и завершить все открытые потоки. Для завершения клиентских сокетов, необходимо выйти из цикла `while`. Поэтому для завершения сервера закрывается слушающий сокет, который как раз доступен из главного потока сервера, а затем начинаем ожидать завершения прослушивающего потока. После выхода из цикла прослушивания (так как из главного потока слушающий сокет был закрыт), закрываем все клиентские сокеты и ожидаем завершения потоков серверов. Так как произошло закрытие клиентских сокетов, то в потоке чтения и выполнения клиентских команд получается ошибка при выполнении команды `readn`, что приводит к выходу из цикла и завершению потока. Таким образом, после завершения всех клиентских потоков, завершается слушающий поток, а также происходит выход из `while` в главном потоке сервера, где производится финальная команда `WSACleanup()`;

На сервере определено 2 специальные функции считывания и отправки данных: `readn` и `sendn`. Первая гарантированно считывает `n` байт следующим образом: в цикле контролируем количество реально принятых байт (`k`), затем в передвигаем указатель `y` буфера на `k` и уменьшаем общее число еще несчитанных байт (`nLeft`) на `k`. Вторая - посылает сообщение, длина (`n`) которого больше длины сообщения, которая задана в протоколе. Так посылает первую посылку длиной равной протокольной длине и увеличиваем количество отосланных байт на это количество. Повторяем посылки в цикле, передвигая указатель в буфере на протокольную длину и увеличивая количество отосланных байт. В конце концов, количество отосланных байт станет больше или равно заданному количеству байт `n`. Отправка завершится.

## 6.7. Описание архитектуры и особенности реализации UDP

Основная логика взаимодействия клиента и сервера осталась такой же, как у TCP. Однако в данном случае нет необходимости создавать отдельные сокеты и потоки для каждого клиента. Обработка всех запросов происходит в основном потоке. Со стороны

клиента функция приема сообщения от сервера также была перенесена в основной поток. Для отправки и приема сообщений написаны функции `sendOneTime` и `recvOneTime`. Данные функции необходимы для осуществления контроля целостности данных. Функция `sendOneTime` перед отправкой сообщения вставляет в начало пакета его порядковый номер, а `recvOneTime` осуществляет проверку этого номера и вывод ошибки в случае несовпадения фактического номера пакета с ожидаемым.

## 7. Дополнительное задание

Исследуем реальные прикладные протоколы. Притворимся клиентом и подключимся к одному из существующих общедоступных серверов.

### 7.1. Подключение к HTTP-серверу и запрос веб-страницы

С помощью команды `telnet` подключаемся к HTTP-серверу:

```
1 telnet tproger.ru 80
```

`tproger.ru` - подключаемый сервер, `80` - номер порта.

Затем с помощью команды запрашиваем определенную веб-страницу:

```
1 GET / HTTP/1.1  
2 HOST: tproger.ru
```

Веб-страница успешно выведена (рис.7.1).

```

user@user-VirtualBox:~/Рабочий стол$ telnet tproger.ru 80
Trying 104.24.4.55...
Connected to tproger.ru.
Escape character is '^]'.
GET / HTTP/1.1
HOST: tproger.ru

HTTP/1.1 301 Moved Permanently
Date: Sun, 17 Dec 2017 22:42:03 GMT
Content-Type: text/html
Set-Cookie: cfduid=d7fd8624a59a592f2236de5d4a9db06611513550523; expires=Mon, 17-Dec-18 22:42:03 GMT; path=/; domain=.tproger.ru; HttpOnly
Location: https://tproger.ru/
Strict-Transport-Security: max-age=31536000
X-Xss-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Server: cloudflare-nginx
CF-RAY: 3ced64b275367630-ARN
Transfer-Encoding: chunked
X-OSSPoxy: OSSProxy 1.3.337.407 (Build 337.407 Win32 en-us)(May 24 2017 16:51:00)
Connection: keep-alive

1c3
<html>
<head><title>301 Moved Permanently</title><script type="text/javascript">
//
window._mirage2 = {petok:"eealea62154ca50a491cdd21e5b051b496843aac-1513550523-1800"};
//]]&gt;
&lt;/script&gt;
&lt;script type="text/javascript" src="https://ajax.cloudflare.com/cdn-cgi/scripts/04b3eb47/cloudflare-static/mirage2.min.js"&gt;&lt;/script&gt;
&lt;/head&gt;
&lt;body bgcolor="white"&gt;
&lt;center&gt;&lt;h1&gt;301 Moved Permanently&lt;/h1&gt;&lt;/center&gt;
&lt;hr&gt;&lt;center&gt;nginx&lt;/center&gt;
&lt;/body&gt;
&lt;/html&gt;

0
</pre>
</div>
<div data-bbox="295 445 699 463" data-label="Caption">
<p>Рисунок 7.1. Подключение к HTTP-серверу</p>
</div>
<div data-bbox="90 513 906 560" data-label="Section-Header">
<h2>7.2. Подключение к FTP-серверу, запрос списка файлов в директории и получение файла</h2>
</div>
<div data-bbox="138 574 670 591" data-label="Text">
<p>С помощью команды telnet подключаемся к FTP-серверу:</p>
</div>
<div data-bbox="74 601 366 617" data-label="Text">
<pre>1 telnet ftp.stat.duke.edu 21</pre>
</div>
<div data-bbox="138 635 840 651" data-label="Text">
<p>ftp.stat.duke.edu - ftp-сервер, к которому происходит подключение, 21 - порт.</p>
</div>
<div data-bbox="90 657 907 808" data-label="Text">
<p>Вводим имя пользователя и пароль. Затем необходимо перейти в пассивный режим с помощью команды pasv. С помощью цифр кодируется IP-адрес и порт для соединения. Создаем соединение к указанному IP-адресу через необходимый порт (порт высчитывается по формуле <math>p1 \cdot 256 + p2</math>, где <math>p1</math> и <math>p2</math> - это два последних числа в присланном сообщении сервера). Стоит уточнить, что здесь используется одно соединение на одну команду. С помощью команды pwd и cwd переходим по папкам рис.7.2, пока не найдем необходимый файл. Для скачивания файла используем команду retr &lt;имя файла&gt;.</p>
</div>
<div data-bbox="138 812 428 830" data-label="Text">
<p>Файл успешно скачан (рис.7.3).</p>
</div>
<div data-bbox="485 952 513 969" data-label="Page-Footer">
<p>12</p>
</div>
```

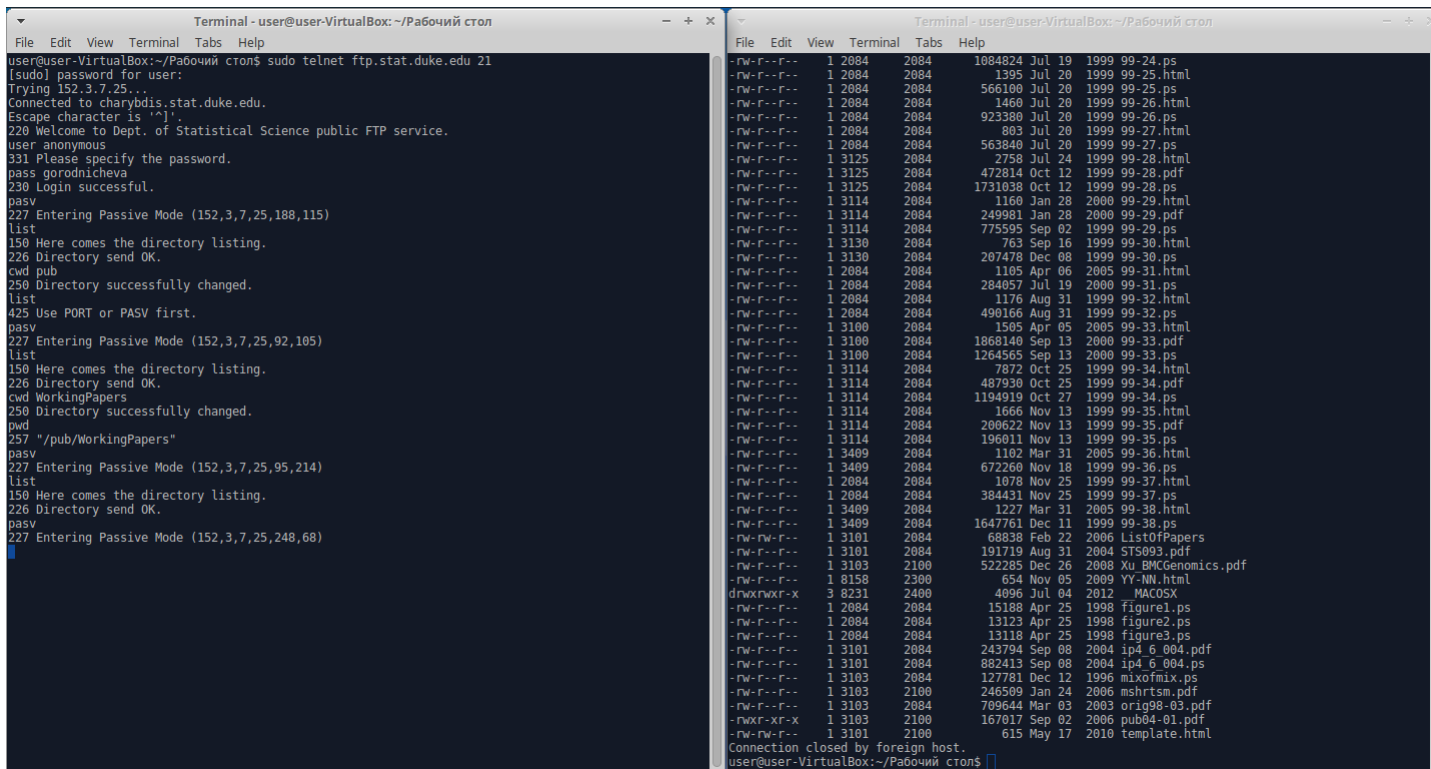


Рисунок 7.2. Подключение к FTP-серверу

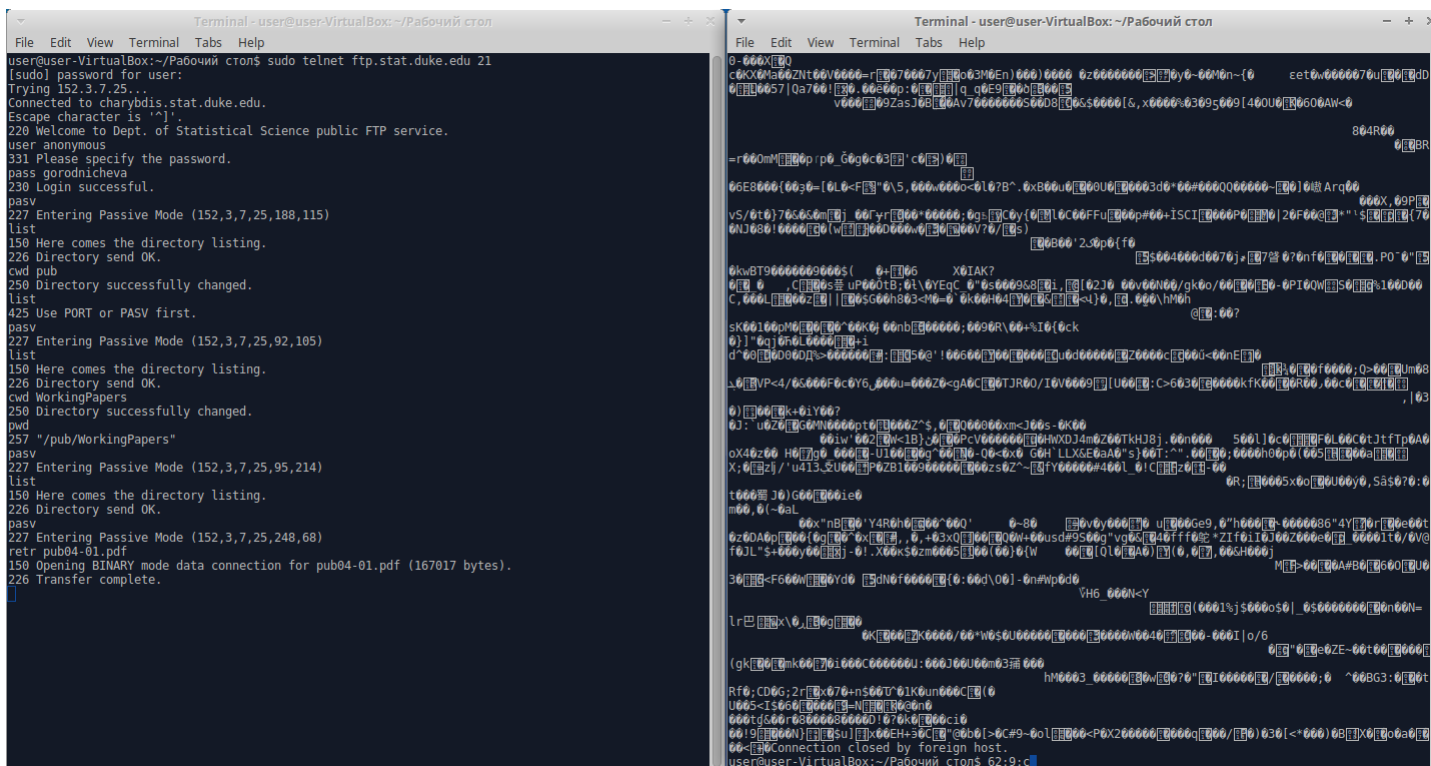



Рисунок 7.3. Скачивание файла с FTP-сервера

### 7.3. Подключение к SMTP-серверу и отправка письма

С помощью команды, указанной ниже, подключаемся к SMTP-серверу smtp.yandex.ru (рис. 7.4):

```
1 gnutils-cli -p 465 smtp.yandex.ru
```



```
user@user-VirtualBox:~/Рабочий стол$ gnutils-cli -p 465 smtp.yandex.ru
Processed 148 CA certificate(s).
Resolving 'smtp.yandex.ru'...
Connecting to '213.180.193.38:465'...
- Certificate type: X.509
- Got a certificate list of 3 certificates.
- Certificate[0] info:
  - subject 'C=RU,O=Yandex LLC,OU=IT0,L=Moscow,ST=Russian Federation,CN=smtp.yand
ex.ru', issuer 'C=RU,O=Yandex LLC,OU=Yandex Certification Authority,CN=Yandex CA
', RSA key 2048 bits, signed using RSA-SHA256, activated '2017-10-11 13:27:26 UT
C', expires '2019-10-11 13:27:26 UTC', SHA-1 fingerprint '57bed6b3bce8e3b8c1a67a
9bc7faf28d480cabb'
  Public Key ID:
    29937c5abb1a07bc8f944012d6092af73f2306b8
  Public key's random art:
  +-[ RSA 2048]-----+
  | +.o |
  | o.o |
  | o.o |
  |.o + o..|
  | . o B S |
  | . o X . |
  | E  o X o |
  | . o B . |
  |      o.o |
  +-----+
- Certificate[1] info:
  - subject 'C=RU,O=Yandex LLC,OU=Yandex Certification Authority,CN=Yandex CA', i
ssuer 'C=PL,O=Unizeto Technologies S.A.,OU=Certum Certification Authority,CN=Cer
tum Trusted Network CA', RSA key 2048 bits, signed using RSA-SHA256, activated '
2015-01-21 12:00:00 UTC', expires '2025-01-18 12:00:00 UTC', SHA-1 fingerprint '
ddf10e6da72c447ecad874eb531b49662d2c6ed2'
- Certificate[2] info:
  - subject 'C=PL,O=Unizeto Technologies S.A.,OU=Certum Certification Authority,C
N=Certum Trusted Network CA', issuer 'C=PL,O=Unizeto Sp. z o.o.,CN=Certum CA', R
SA key 2048 bits, signed using RSA-SHA256, activated '2008-10-22 12:07:37 UTC',
expires '2027-06-10 10:46:39 UTC', SHA-1 fingerprint '929badf26081523490edc91154
b380a4776e2185'
- Status: The certificate is trusted.
- Description: (TLS1.2)-(ECDHE-RSA-SECP256R1)-(AES-128-GCM)
```

Рисунок 7.4. Подключение к SMTP-серверу

Данная команда обеспечивает защищенное TLS-подключение. Команды telnet/nc здесь не подходят по причине их работы только с нешифрованными соединениями.

SSL (Secure Sockets Layer) и TLS (Transport Level Security) - это криптографические протоколы, обеспечивающие защищенную передачу данных в компьютерной сети. Они активно используются при работе с электронной почтой. Соединение, защищенное протоколом TLS, обладает определенными свойствами: безопасностью, аутентификацией и целостностью.

Затем необходимо пройти аутентификацию:

```

220 smtp40.mail.yandex.net ESMTP (Want to use Yandex.Mail for your domain? Visit
http://pdd.yandex.ru)
ehlo yandex.ru
250-smtp40.mail.yandex.net
250-8BITMIME
250-PIPELINING
250-SIZE 42991616
250-AUTH LOGIN PLAIN XOAUTH2
250-DSN
250-ENHANCEDSTATUSCODES
AUTH LOGIN
334 VXNlcm5hbWU6
Z29yb2RuYWNoZXZhLmxpZGhhbmRleC5ydQ==
334 UGFzc3dvcmQ6
Tm9yYTAxMDE=
235 2.7.0 Authentication successful.

```

Рисунок 7.5. Аутентификация

Для аутентификации использовался BASE64, с помощью которого были закодированы логин и пароль (рис. 7.6):

```

user@user-VirtualBox:~/Рабочий стол$ perl -MMIME::Base64 -e 'print encode_base64
("user@yandex.ru");'
Tm9yYTAxMDE=
user@user-VirtualBox:~/Рабочий стол$ perl -MMIME::Base64 -e 'print encode_base64
("gorodnicheva.lidia@yandex.ru");'
Z29yb2RuYWNoZXZhLmxpZGhhbmRleC5ydQ==

```

Рисунок 7.6. Кодирование логина и пароля

Используя команды MAIL FROM (от кого письмо), RCPT TO (кому письмо), DATA (собственно само письмо с указанием темы (Subject) и From (от кого), было отослано письмо с содержанием ‘Gorodnicheva Lidia’ (рис. 7.7):

```

MAIL FROM: gorodnicheva.lidia@yandex.ru
250 2.1.0 <gorodnicheva.lidia@yandex.ru> ok
RCPT TO: gorodnicheva.lidia@gmail.com
250 2.1.5 <gorodnicheva.lidia@gmail.com> recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Hello!
From: gorodnicheva.lidia@yandex.ru

Gorodnicheva Lidia
.
250 2.0.0 Ok: queued on smtp1j.mail.yandex.net as 1513449839-nu1P1ijJwp-fko46bIL

```

Рисунок 7.7. Посылка письма

## 7.4. Подключение к POP3-серверу, проверка почты и получение письма

Попробуем прочитать посланное письмо с помощью протокола POP3. Для начала подключимся к POP3-серверу (рис. 7.8):



```

user@user-VirtualBox:~/Рабочий стол$ gnutls-cli -p 995 pop.gmail.com
Processed 148 CA certificate(s).
Resolving 'pop.gmail.com'...
Connecting to '74.125.131.108:995'...
- Certificate type: X.509
- Got a certificate list of 2 certificates.
- Certificate[0] info:
- subject 'C=US,ST=California,L=Mountain View,O=Google Inc,CN=pop.gmail.com', issuer 'C=US,O=Google Trust Services,CN=Google Internet Authority G3', expires '2017-12-05 10:17:36 UTC', expires '2018-02-27 09:29:00 UTC', SHA-1 fingerprint '8381b0d5c8e0465200ba18aeac9adcafeb55a518'
Public Key ID:
0c98b89e9ace51eee9c4cf7e98b940d127bcc371
Public key's random art:
+---[ RSA 2048]-----+
|
| .oo
| ..o=E
| .o *o
| .o + S
| .=. .
| .o= +
| .o+ =+ .
| +o.+.=o
|
+-----+
- Certificate[1] info:
- subject 'C=US,O=Google Trust Services,CN=Google Internet Authority G3', issuer 'OU=GlobalSign Root CA - R2,O=GlobalSign,CN=GlobalSign', RSA key expires '2021-12-15 00:00:42 UTC', expires '2021-12-15 00:00:42 UTC', SHA-1 fingerprint 'eeacbd0cb452819577911e1e6203db262f84a318'
- Status: The certificate is trusted.
- Description: (TLS1.2)-(ECDHE-RSA-SECP256R1)-(AES-128-GCM)
- Session ID: 6E:2B:C0:97:96:37:D4:9C:B1:DC:03:31:7F:F3:B4:63:FD:80:64:F2:B4:46:27:C4:98:DD:81:7E:DF:BD:70:DB
- Ephemeral EC Diffie-Hellman parameters
- Using curve: SECP256R1
- Curve size: 256 bits
- Version: TLS1.2
- Key Exchange: ECDHE-RSA
- Server Signature: RSA-SHA256
- Cipher: AES-128-GCM
- MAC: AEAD
- Compression: NULL
- Options: extended master secret, safe renegotiation,
- Handshake was completed

- Simple Client Mode:

+OK Gpop ready for requests from 89.223.47.218 100mb317204034lfp

```

Рисунок 7.8. Подключение к POP3-серверу

Затем введем логин и пароль, используя команды USER и PASS (рис. 7.9):

```

+OK Gpop ready for requests from 89.223.47.218 x18mb28028886lta
USER gorodnicheva.lidia@gmail.com
+OK send PASS
PASS *****
+OK Welcome.

```

Рисунок 7.9. Аутентификация

С помощью команды LIST запрошен список всем писем (рис. 7.10), а команда RETR <номер письма> вывела полученное письмо (рис. 7.11):

```

183 110477
184 110743
185 661712
186 110135
187 117828
188 111181
189 111071
190 13389
191 109674
192 119857
193 108577
194 107812
195 110106
196 19159
197 4568
.
RETR 197
+OK message follows
Delivered-To: gorodnicheva.lidia@gmail.com
Received: by 10.140.39.41 with SMTP id u38csp928127qgu;
        Sat, 16 Dec 2017 10:44:00 -0800 (PST)

```

Рисунок 7.10. Список писем

```

Received: by smtp1j.mail.yandex.net (nwsmtplj/Yandex) with ESMTPSA id nu1PlijJwp-f
ko46bIL;
        Sat, 16 Dec 2017 21:42:21 +0300
        (using TLSv1.2 with cipher ECDHE-RSA-AES128-GCM-SHA256 (128/128 bits))
        (Client certificate not present)
Message-Id: <20171216214359.fko46bIL@smtp1j.mail.yandex.net>
Date: Sat, 16 Dec 2017 21:43:59 +0300
To: undisclosed-recipients:;
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=yandex.ru; s=mail; t=151
3449839;
        bh=xbSMXELUKSLW8Dh8LMDvmlWAhrTMe3A14/v9aHN4Rp/g=; h=Subject:From;
        b=oo/J373Ln/hUjy3IbU3wow5t3h+uhMZEoSTJFwmhlg8kGVYJrJoMLIp5v8cRhCr38
        rKdL+0lr7BaFuIAIdYR/jFgUsgJtdHsmW0B/Hk0KIv8oEsfYhwWF9Q5jXn+bKGLTG
        qaWJSEET2KYCx2Inq/RSaw1rq6oLPw2MGJSG/Vjs=
Authentication-Results: smtp1j.mail.yandex.net; dkim=pass header.i=@yandex.ru
Subject: Hello!
From: gorodnicheva.lidia@yandex.ru
Gorodnicheva Lidia
.

```

Рисунок 7.11. Полученное письмо

## 8. Выводы

В данной лабораторной работе изучены принципы программирования сокетов на основе TCP и UDP. Познакомились с архитектурой разработки многопоточных сетевых приложений: использование механизма нитей и средств синхронизации. Изучение происходило через разработку простейших клиент-серверных приложений на ОС Linux и Windows, а также реализацию собственного протокола на основе TCP и UDP для индивидуального задания (а именно дистанционного тестирования). Чтобы не усложнять задание и не создавать отдельную базу данных для пользователей и их результатов, были созданы отдельные текстовые файлы, в которых эти пользователи хранятся. Для вопросов и вариантов ответов также есть свои файлы. Архитектура TCP приложения была построена на основе 3 различных потоков (один поток для приема (read), один поток для подтверждения соединения и создания новых сокетов (accept), один поток для обработки команд с серверной консоли), обеспечивая тем самым параллельную работу всей системы. Минус

такой архитектуры в том, что для каждого клиента выделяется отдельный поток, что получается весьма ресурсоемко. Решить такую проблему можно, например, при помощи пула сокетов (или портов завершения, которые имеются в Windows). Передача сообщений по TCP тоже реализована довольно просто: мы передавали определенное количество байт с каждой посылкой, а завершали сообщение меткой.

Для передачи сообщений длиной больше, чем длина, заданная в протоколе, была создана специальная функция, которая делит сообщение на куски заданной длины и передает их поочередно.

При разработке UDP при попытке создавать новые сокет для клиентов получали ошибку соединения, в результате чего принято решение использовать один сокет для обмена сообщениями. Чтобы понимать, что все посылки доставляются верно или неверно, инкапсулировали номер посылки в сообщение. Если номер не совпадал с ожидаемым, просто выдавали ошибку. Здесь для сохранения длины посылки полную длину посылки оставили той же, однако количество байт данных в ней каждый раз меняется, в зависимости от длины номера посылки. На приеме просто отделяем номер посылки.

Также были получены навыки работы с такими прикладными протоколами, как HTTP, FTP, SMTP и POP3. Эти навыки могут оказаться полезными при разработке серьезных сетевых приложений. Некоторые задачи данной лабораторной очень схожи с задачами, решаемыми этими протоколами. Так SMTP ожидает ответа на каждую операцию и присылает обратно код - выполнена ли операция или завершилась с какой-либо ошибкой. В нашем задании незарегистрированный и незалогиненый пользователь не может получить и пройти тест. В POP3, SMTP, а так же FTP перед тем, как выполнить какую-либо операцию, необходимо было произвести аутентификацию.

## 9. Приложение

Листинг 7: TCP сервер на Linux

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <netdb.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
7
8 #include <string.h>
9
10
11 int readn(int sockfd, char *buf, int n){
12     int k;
13     int off = 0;
14     for(int i = 0; i < n; ++i){
```

```

14         k = read(sockfd, buf + off, 1);
15         off += 1;
16         if (k < 0) {
17             printf("Error: _reading_from_socket_\n");
18             exit(1);
19         }
20     }
21     return off;
22 }
23
24 int main(int argc, char *argv[]) {
25     int sockfd, newsockfd;
26     uint16_t portno;
27     unsigned int clilen;
28     char buffer[256];
29     char *p = buffer;
30     struct sockaddr_in serv_addr, cli_addr;
31     ssize_t n;
32
33     sockfd = socket(AF_INET, SOCK_STREAM, 0);
34
35     if (sockfd < 0) {
36         perror("ERROR: opening_socket");
37         exit(1);
38     }
39
40     bzero((char *) &serv_addr, sizeof(serv_addr));
41     portno = 5001;
42
43     serv_addr.sin_family = AF_INET;
44     serv_addr.sin_addr.s_addr = INADDR_ANY;
45     serv_addr.sin_port = htons(portno);
46
47     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
48         perror("ERROR: on_binding");
49         exit(1);
50     }
51
52     listen(sockfd, 5);
53     clilen = sizeof(cli_addr);
54
55     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
56
57     shutdown(sockfd, 2);
58     close(sockfd);
59
60     if (newsockfd < 0) {

```

```

61     perror("ERROR_on_accept");
62     exit(1);
63 }
66
66     bzero(buffer, 256);
68
68     n = readn(newsockfd, p, 255);
70
70     printf("Here_is_the_message:_%s\n", buffer);
72
72     n = write(newsockfd, "I_got_your_message", 18);
74
74     if (n < 0) {
75         perror("ERROR_writing_to_socket");
76         exit(1);
77     }
79
79     shutdown(newsockfd, 2);
80     close(newsockfd);
82
82     return 0;
83 }

```

#### Листинг 8: TCP клиент на Linux

```

1 #include <stdio.h>
2 #include <stdlib.h>
4
4 #include <netdb.h>
5 #include <netinet/in.h>
6 #include <unistd.h>
8
8 #include <string.h>
10
10 int readn(int sockfd, char *buf, int n){
11     int k;
12     int off = 0;
13     for(int i = 0; i < n; ++i){
14         k = read(sockfd, buf + off, 1);
15         off += 1;
16         if (k < 0){
17             printf("Error_reading_from_socket_\n");
18             exit(1);
19         }
20     }
21     return off;

```

```

22 }
24
24 int main(int argc, char *argv[]) {
25     int sockfd, n;
26     uint16_t portno;
27     struct sockaddr_in serv_addr;
28     struct hostent *server;
29
30
30     char buffer[256];
31     char *p = buffer;
32
33
33     if (argc < 3) {
34         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
35         exit(0);
36     }
37
38     portno = (uint16_t) atoi(argv[2]);
39
40     sockfd = socket(AF_INET, SOCK_STREAM, 0);
41
42
42     if (sockfd < 0) {
43         perror("ERROR_opening_socket");
44         exit(1);
45     }
46
47     server = gethostbyname(argv[1]);
48
49
49     if (server == NULL) {
50         fprintf(stderr, "ERROR_no_such_host\n");
51         exit(0);
52     }
53
54
54     bzero((char *) &serv_addr, sizeof(serv_addr));
55     serv_addr.sin_family = AF_INET;
56     bcopy(server->h_addr, (char *) &serv_addr.sin_addr.s_addr, (size_t) server->
↪ h_length);
57     serv_addr.sin_port = htons(portno);
58
59
59     if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
↪ {
60         perror("ERROR_connecting");
61         exit(1);
62     }
63
64
65
65     printf("Please_enter_the_message:_");
66     bzero(buffer, 256);

```

```

67     fgets(buffer, 255, stdin);
69
69     n = write(sockfd, buffer, 255);
71
71     if (n < 0) {
72         perror("ERROR_writing_to_socket");
73         exit(1);
74     }
76
76     bzero(buffer, 256);
77     n = readn(sockfd, p, 255);
79
79     printf("%s\n", buffer);
81
81     shutdown(sockfd, 2);
82     close(sockfd);
84
84     return 0;
85 }

```

Листинг 9: UDP сервер на Windows

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <winsock2.h>
5 #include <stdint.h>
8
8
8 int main() {
10
10     WSADATA wsaData;
12
12     unsigned int t;
13     t = WSASStartup(MAKEWORD(2,2), &wsaData);
15
15     if (t != 0) {
16         printf("WSAStartup_failed:_%ui\n", t);
17         return 1;
18     }
20
20     int sockfd;
21     uint16_t portno;
22     int clilen;
23     char buffer[256];
24     struct sockaddr_in serv_addr, cli_addr;
26

```

```

26  /* First call to socket() function */
27  sockfd = socket(AF_INET, SOCK_DGRAM, 0);
29
30  if (sockfd < 0) {
31      perror("ERROR_opening_socket");
32      exit(1);
33  }
34
35  /* Initialize socket structure */
36  memset((char *) &serv_addr, 0, sizeof(serv_addr));
37  portno = 5001;
38
39  serv_addr.sin_family = AF_INET;
40  serv_addr.sin_addr.s_addr = INADDR_ANY;
41  serv_addr.sin_port = htons(portno);
42
43  /* Now bind the host address using bind() call.*/
44  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
45      perror("ERROR_on_binding");
46      exit(1);
47  }
48
49  clilen = sizeof(cli_addr);
50  int rec = recvfrom(sockfd, buffer, 256, 0, (struct sockaddr *)&cli_addr, &
↪ clilen);
51  if (rec < 0) {
52      perror("ERROR");
53      exit(1);
54  }
55
56  struct hostent *hst;
57  hst = gethostbyaddr((char *)&cli_addr.sin_addr, 4, AF_INET);
58  printf("+%s_[%s:%d]_new_DATAGRAM!\n", (hst) ? hst->h_name : "Unknown_
↪ host", inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));
59
60  printf("C=>S:%s\n", &buffer[0]);
61
62  sendto(sockfd, buffer, 256, 0, (struct sockaddr *)&cli_addr, sizeof(
↪ cli_addr));
63  close(sockfd);
64  WSACleanup();
65
66  return 0;
67 }

```

Листинг 10: UDP клиент на Windows



```

1 #include <stdio.h>
2 #include <winsock2.h>
3 #include <stdint.h>
4
5
6
6 int main(int argc, char *argv[]) {
7
8     WSADATA wsaData;
9
10
10     unsigned int t;
11     t = WSStartup(MAKEWORD(2,2), &wsaData);
12
13
13     if (t != 0) {
14         printf("WSAStartup failed: %ui\n", t);
15         return 1;
16     }
17
18
18     int sockfd, n;
19     uint16_t portno;
20     struct sockaddr_in serv_addr;
21     struct hostent *server;
22
23
23     char buffer[256];
24
25
25     if (argc < 3) {
26         fprintf(stderr, "usage: %s hostname port\n", argv[0]);
27         exit(0);
28     }
29
30
30     portno = (uint16_t) atoi(argv[2]);
31
32
32     /* Create a socket point */
33     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
34
35
35     if (sockfd < 0) {
36         perror("ERROR opening socket");
37         exit(1);
38     }
39
40
40     server = gethostbyname(argv[1]);
41
42
42     if (server == NULL) {
43         fprintf(stderr, "ERROR, no such host\n");
44         exit(0);
45     }
46
47
47     memset((char *) &serv_addr, 0, sizeof(serv_addr));

```

```

48     serv_addr.sin_family = AF_INET;
49     memcpy((char *) &serv_addr.sin_addr.s_addr, server->h_addr, (size_t) server
↪->h_length);
50     serv_addr.sin_port = htons(portno);
52
52     printf("Please_enter_the_message:_");
53     memset(buffer, 0, 256);
54     fgets(buffer, 255, stdin);
56
56     /* Send message to the server */
57     n = sendto(sockfd, buffer, 256, 0, (struct sockaddr *)&serv_addr, sizeof(
↪serv_addr));
59
59     if (n < 0) {
60         perror("ERROR_writing_to_socket");
61         exit(1);
62     }
64
64     /* Now read server response */
65     memset(buffer, 0, 256);
67
67     struct sockaddr_in serv_two;
68     int serv_two_size = sizeof(serv_two);
70
70     int rec = recvfrom(sockfd, buffer, 256, 0, (struct sockaddr *)&serv_two, &
↪serv_two_size);
71     if (rec < 0) {
72         perror("ERROR");
73         exit(1);
74     }
76
76     printf("%s\n", buffer);
78
78     closesocket(sockfd);
79     WSACleanup();
81
81     return 0;
82 }

```

Листинг 11: Многопоточный TCP-сервер на Linux

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <netdb.h>
4 #include <netinet/in.h>
5 #include <unistd.h>
7

```

```

7 #include <string.h>
8 #include "pthread.h"
9
10
11 int readn(int sockfd, char *buf, int n){
12     int k;
13     int off = 0;
14     for(int i = 0; i < n; ++i){
15         k = read(sockfd, buf + off, 1);
16         off += 1;
17         if (k < 0){
18             printf("Error._reading_from_socket_\n");
19             exit(1);
20         }
21     }
22     return off;
23 }
24
25 void* formulti (void* temp) {
26     int n = *((int*)temp);
27     char buffer[256];
28     char *p = buffer;
29     bzero(buffer, 256);
30     int k = readn (n, p, 255);
31     if (k > 0) {
32         printf("%s_\n", buffer);
33         write(n, "I_have_got_your_message", 23);
34     }
35     shutdown(n, 2);
36     close(n);
37 }
38
39
40
41 int main() {
42     int sockfd, newsockfd;
43     uint16_t portno;
44     unsigned int clilen;
45     char buffer[256];
46     struct sockaddr_in serv_addr, cli_addr;
47     ssize_t n;
48
49     sockfd = socket(AF_INET, SOCK_STREAM, 0);
50
51     if (sockfd < 0) {
52         perror("ERROR_opening_socket");
53         exit(1);
54     }

```

```

55
55     bzero((char *) &serv_addr, sizeof(serv_addr));
56     portno = 5001;
57
58     serv_addr.sin_family = AF_INET;
59     serv_addr.sin_addr.s_addr = INADDR_ANY;
60     serv_addr.sin_port = htons(portno);
61
62     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
63         perror("ERROR_on_binding");
64         exit(1);
65     }
66
67     while (1) {
68         listen(sockfd, 5);
69         clilen = sizeof(cli_addr);
70
71         newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
72
73         if (newsockfd < 0) {
74             perror("ERROR_on_accept");
75             exit(1);
76         }
77
78         pthread_t tid;
79         pthread_attr_t attr;
80         pthread_attr_init(&attr);
81         pthread_create(&tid, &attr, formulti, &newsockfd);
82         pthread_detach(tid);
83     }
84
85     return 0;
86 }

```