

Сети ЭВМ и телекоммуникации

А. В. Никитина

19 декабря 2014 г.

Глава 1

Система дистанционного тестирования

1.1 Функциональные требования

1.1.1 Задание

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

1.1.2 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

1.1.3 Настройки приложений

Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени удалённого сервера тестов и номера порта, используемого сервером. Разработанное серверное приложение должно хранить вопросы и правильные ответы нескольких тестов.

1.2 Нефункциональные требования

1.2.1 Требования к реализации

Соединение начинает клиент, отправляя серверу фиксированную строку, говорящую о начале соединения (пусть, например, это строка "!"). Далее происходит обмен необходимым набором сообщений: регистрация клиента, выбор теста, прохождение теста. После получения результата клиент разрывает соединение с сервером.

1.2.2 Требования к надежности

Длина отправляемого пакета от клиента серверу при регистрации клиента должна проверяться на максимальное значение, так мы защищаем сервер от "падения" при отправке слишком длинного сообщения.

При отправке от клиента серверу ответа на вопрос или различных запросов формируем пакеты длиной=1, так как больше нам и не требуется.

1.3 Накладываемые ограничения

- **Ограничения на длину пакетов.** Все запросы клиента (на соединение, на получение списка теста, на получение очередного вопроса), а также ответы клиента имеют длину 1 символ.

Пусть остальные пакеты TCP будут иметь максимальный размер 512 символов. Размер выбран не случайно.

MSS (Maximum segment size) является параметром протокола TCP и определяет максимальный размер полезного блока данных в байтах для TCP пакета (сегмента). Таким образом этот параметр не учитывает длину заголовков TCP и IP. Для установления корректной TCP сессии с удалённым хостом должно соблюдаться следующее условие:

$$MSS + TCP + IP \leq MTU$$

Таким образом, максимальный размер $MSS = MTU - \text{размер заголовка IPv4} - \text{размер заголовка TCP}$

Так каждый хост на IPv4 требует доступности для MSS последних 536 октетов ($= 576 - 20 - 20$) а на IPv6 — 1220 октетов ($= 1280 - 40 - 20$).

Мы выбрали значение, являющееся степенью 2 - 512. Это является оптимальным размером пакетов.

UDP передает пакеты отдельными датаграммами (заголовок и данные) в байтах. Таким образом, если длина пакета с UDP будет превышать MTU (для Ethernet по умолчанию 1500 байт), то отправка такого пакета может вызвать его фрагментацию, что может привести к тому, что он вообще не сможет быть доставлен, если промежуточные маршрутизаторы или конечный хост не будут поддерживать фрагментированные IP пакеты. Также в RFC791 указывается минимальная длина IP пакета не менее 576 байт и рекомендуется отправлять IP пакеты большего размера только в том случае если вы уверены, что принимающая сторона может принять пакеты такого размера. Следовательно, чтобы избежать фрагментации UDP

пакетов (и возможной их потери), размер данных в UDP не должен превышать: $MTU - (Max\ IP\ Header\ Size) - (UDP\ Header\ Size) = 1500 - 60 - 8 = 1432$ байт. Для того чтобы быть уверенным, что пакет будет принят любым хостом, размер данных в UDP не должен превышать: (минимальная длина IP пакета) $- (Max\ IP\ Header\ Size) - (UDP\ Header\ Size) = 576 - 60 - 8 = 508$ байт.

Для надежности, зададим максимальную размерность отправляемых пакетов 500 байт.

- **Обрыв сессии.** При обрыве сессии, то есть при непрохождении всего цикла протокола, все результаты, введенные на начальных этапах теряются. Это является минусом протокола.

Глава 2

Реализация для работы по протоколу TSP

2.1 Прикладной протокол

Имя	Формат	Действие	Длина
ECHO-REQUEST	!	клиент проверяет наличие сервера в сети	1
ECHO-ANSWER	!	сервер отвечает клиенту о своем присутствии в сети	1
LOGIN-REQUEST	login	клиент посылает свой логин серверу	strlen(login)
LOGIN-ANSWER	str 1+2+3+login/	сервер отвечает клиенту о регистрации в сети	strlen(str)
TESTS-REQUEST	1	клиент запрашивает список тестов	1
TESTS-ANSWER	str1 1+2+3.../	сервер выдает клиенту список тестов	strlen(str1)
CHOICE-TEST	numb	клиент отправляет номер теста	strlen(numb)
CHOICE-OK	1	сервер подтверждает получение номера теста	1
QUEST-REQUEST	2	клиент запрашивает наличие следующего вопрос	1
QUEST-YES	2	сервер отвечает что следующий вопрос есть	1
QUEST-NO	/	сервер отвечает что следующего вопроса нет	1
QUESTION	quest	сервер отправляет вопрос клиенту	strlen(quest)
ANSWER	answer= (1,2,3,4)	клиент отправляет ответ серверу	1
TRUE	true= (Right, Wrong) ⁶	сервер отправляет клиенту правильность ответа	6
RESULT	result 1+2+3+login/	сервер отправляет клиенту его результат	strlen(result)

Формат строки **str**

1+2+3+login/

+ - знак-разделитель, разделяет элементы строки

/ - знак конца строки

Элементы строки в порядке их следования:

1 - номер последнего пройденного теста

2 - количество вопросов в тесте

3 - количество верных ответов

login - логин пользователя

Формат строки **str1**

1+2+3.../

+ - знак-разделитель, разделяет элементы строки

/ - знак конца строки

Элементы строки :

1, 2, 3 - все имеющиеся номера тестов

Формат строки **result**

1+2+3+login/

+ - знак-разделитель, разделяет элементы строки

/ - знак конца строки

Элементы строки в порядке их следования:

1 - номер пройденного теста

2 - количество вопросов в тесте

3 - количество верных ответов

login - логин пользователя

Формат строки **quest**

How are you?+ok+bad+nice+good/

+ - знак-разделитель, разделяет элементы строки

/ - знак конца строки

Элементы строки в порядке их следования:

How are you - вопрос

ok - ответ 1

bad - ответ 2

nice- ответ 3

good- ответ 4

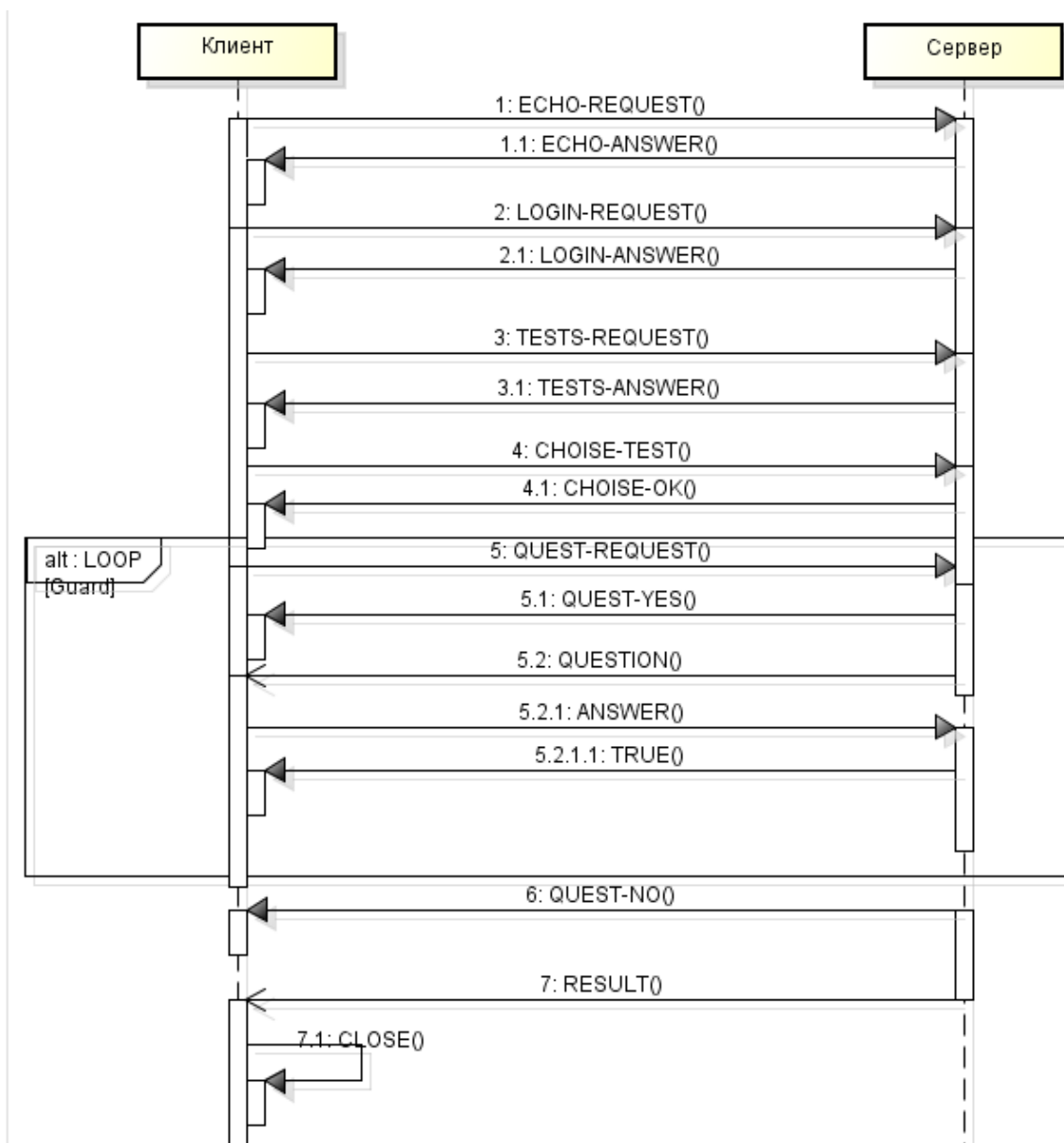


Рис. 2.1: Sequence Diagram.

2.2 Архитектура приложения

2.2.1 Дизайн протокола

После запуска сервер начинает прослушивать определенный порт, он ждет подключение от клиента.

- Клиент отправляет сообщения длиной 1 символ, он может подключиться к серверу, отправив сообщение '!'.
1
- После принятия этого символа, сервер разрешает клиенту передачу смысловых сообщений, отправив в ответ также сообщение длиной 1 символ '!'.
1
- На стороне клиента появляется сообщение

Please enter login:

Клиент вводит свой логин

- После принятия сервер его анализирует, если логина нет в файле registration.txt (файл, хранящий сохраненных пользователей, его формат описан ниже), создается новый пользователь, после чего клиенту отправляется сообщение (формат рассмотрен выше)

1+2+3+login/

Если клиент только что зарегистрировался, клиент оповещает его сообщением

Welocome, login.

Иначе показывает результат последнего тестирования в виде строки

login, your last test is 1. True answers is 2 of 3 answers.

- Клиент отправляет серверу запрос на получение списка тестов (строка '1')
- Сервер в ответ отправляет список тестов в виде, например, такой строки(формат рассмотрен выше)

1+2+3+12/

Клиент выводит у себя список тестов.

List of test: 1 2 3 12

- Клиент выбирает номер теста, имеющегося в папке с тестами
- Сервер отвечает подтверждением принятия номера (строка '1')
- Начинается цикл прохождения теста:
 1. Клиент отправляет запрос на получение следующего вопроса (строка '2', длиной 1 символ)
 2. Сервер передает строку, подтверждающую наличие вопроса (строка '2', длиной 1 символ)
 3. Сервер передает строку, содержащую вопрос и варианты ответов в виде

How are you?+ok+bad+nice+good/

Клиент выводит её у себя в виде

How are you? 1)ok 2)bad 3)nice 4)good

4. Клиент отвечает на вопрос (цифра от 1 до 4)
5. Сервер отправляет "right" либо "wrong" в зависимости от правильности ответа. Клиент выводит эту строку.
Сам сервер на каждый вопрос формирует у себя строку, например "Answer: 1".

Так обрабатываются все вопросы теста.

- После того, как клиент ответит на все вопросы выбранного теста, сервер отправляет ему строку, указывающую на окончание прохождения теста (строка '/', длиной 1 символ).
Модифицирует файл registration.txt, записывая в строку соответствующего пользователя номер пройденного теста, количество вопросов теста и верных ответов клиента.
- Сервер выдает клиенту его результат, например такой.

1+2+3+login/

Клиент выводит ее у себя в виде

The end.

Login, test is 1. The number of true answer is 2, the number of question is 3.

- Клиент обрывает соединение функцией close(s).

Возможно подключение и работа сервера с несколькими клиентами, при этом создается на каждого клиента свой поток, потоки работают параллельно и не влияют на работу других клиентов.

При подключении очередного клиента, сервер выводит у себя сообщение

Number of clients: [number]. Accepted client id: [id]

где [number]-количество клиентов, подключенных к серверу на данный момент, [id]-уникальный идентификатор клиента.

Сервер в любой момент может отключить подключившегося к нему клиента. Для этого он должен отправить команду

k[id]

где [id]-идентификатор клиента.

2.2.2 Форматы файлов и строк

Формат исходных файлов.

Каждый тест хранится в отдельном текстовом файле с именем, соответствующим номеру тесту(1.txt, 2.txt). Каждая строка в этих файлах представляет из себя отдельный вопрос теста.

Файл registration.txt хранит данные зарегистрированных пользователей. Каждая строка соответствует отдельному зарегистрированному пользователю.

Формат строки файла registration.txt

1+2+3+login/

'+'-знак-разделитель, разделяет элементы строки

'/'-знак конца строки

Элементы строки в порядке их следования:

- номер теста
- количество вопросов теста
- количество верных ответов
- логин пользователя

Таким образом, сервер считывает информацию о зарегистрировавшихся пользователях. При считывании строка должна обрабатываться специальным парсером, разбирающим строку соответствующим образом. Создадим новый модуль `registration`, занимающийся парсером строки. Модуль определит структуру читаемого вопроса `struct Client`. Структура состоит из

- `int numberTest`-номер теста
- `int sizeQuestion`-количество вопросов теста
- `int sizeTrueAnswer`-количество верных ответов
- `char *login`-логин пользователя
- `int sizes[4]`- массив с размерами вышеописанных элементов

Также модуль имеет функции для считывания строк и заполнения структур

```
void writeClient (struct Client *c,char *str);
void writeSizeClient(struct Client *c,char *str);
```

При регистрации нового пользователя вызывается функция

```
void new(char *login,struct Client *c);
```

Функция создает нового пользователя, у которого первые 3 элемента =0.

```
0+0+0+login/
```

При прохождении очередного теста, строка, соответствующая пользователю, модифицируется в зависимости от результатов теста с помощью функции

```
void newResult(struct Client *c,int number,int testsize,int
               numberTrueAnswer);
```

Формат строки файлов с тестами

1+How are you?+ok+bad+!nice+good

'+'-знак-разделитель, разделяет элементы строки

'/'-знак конец строки

'!'-знак, обозначающий правильный ответ, ставится перед верным ответом после символа-разделителя '±'

Элементы строки в порядке их следования:

- номер вопроса
- вопрос
- ответ1
- ответ2
- ответ3
- ответ4

Таким образом, сервер для каждого теста должен считывать информацию из нужного файла. При считывании строка должна обрабатываться специальным парсером, разбирающим строку соответствующим образом. Создадим новый модуль `writeStruct`, занимающийся парсером строки. Модуль определит структуру читаемого вопроса `struct Line`. Структура состоит из

- `int number`-номер вопроса
- `char *question`-строка-вопрос
- `char **answer`-массив строк-ответов
- `sizeNumber`-количество цифр в номере вопроса
- `*sizeAnswer`-количество символов в строках-ответах
- `sizeQuest`-количество символов в строке-вопросе
- `trueAnswer`-верный ответ

Также модуль имеет функции для считывания строк и заполнения структур

```
void writeStruct (struct Line *x, char *str);  
void writeSize(struct Line *x,char *str);
```

2.3 Тестирование

2.3.1 Описание тестового стенда и методики тестирования

Для тестирования приложения запустим сервер на ОС Linux и несколько клиентов на ОС Windows. В процессе тестирования проверим основные возможности сервера по параллельному тестированию нескольких пользователей, указанные в заданиях, .

2.3.2 Тестовый план и результаты тестирования

Запустим сервер на ОС Linux, подключимся к серверу, запустив клиента с ОС Windows.

Клиент просит ввести символ '!' для подключения к серверу, при неверном символе, ничего не происходит, вводим '!'.
Клиент просит ввести логин, вводим логин "ann". Видим последний результат этого пользователя.

Клиент просит ввести символ '1' для получения списка тестов, при неверном символе, ничего не происходит, вводим '1'.

Получаем список тестов. Выбираем тест из имеющихся в списке.
Клиент просит ввести символ '2' для получения следующего вопроса, при неверном символе, ничего не происходит, вводим '2'.

Получаем вопрос, вводим ответ (цифра от 1 до 4). Получаем от сервера ответ Wrong, Right.

При окончании вопросов, выводим результат тестирования.

Результат работы приведен на рисунке ниже.

Протестируем механизм многопоточности

После запуска сервера запускаем клиента, сервер выводит на экране информацию о количестве подключенных клиентов и id только что подключившегося. Запустим еще одного клиента.

Для завершения соединения с клиентом сервер должен ввести команду k[id], где id соответствует id клиента, которого мы хотим отсоединить.

После закрытия соответствующего сокета клиента, попытаемся произвести какие-то действия с клиентом, он выведет ошибку и завершит свое выполнение. В то время, как незакрытый клиент продолжает верно выполняться.

Результат работы приведен на рисунке ниже.

```

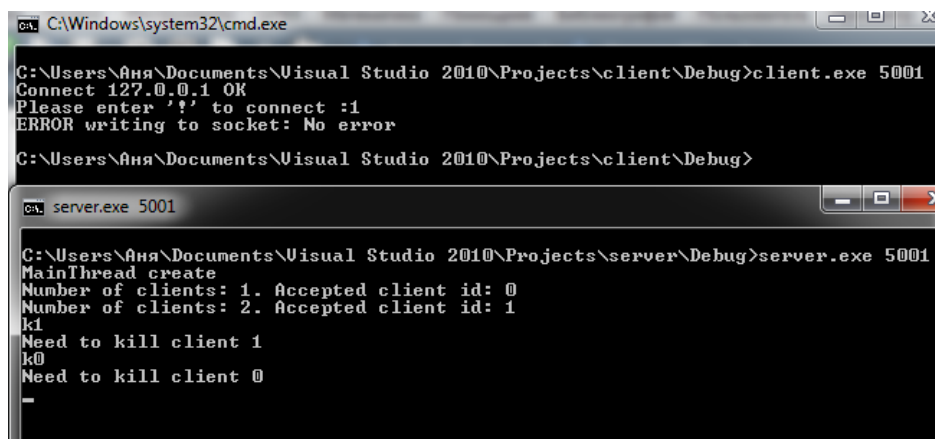
localhost
Connect localhost OK
Please enter '!' to connect :1
Please enter '!' to connect :!
Enter login: ann
ann your last test is 1. Result 0 of 2.
Please enter '1' to receive list of test :2
Please enter '1' to receive list of test :1
List of test: 1 2
Choose test: 3
Choose test: 1
Please enter '2' to next question :2
2+3?1>5 2>6 3>2 4>1
6
Enter your answer(1,2,3,4):
1
Wrong

Please enter '2' to next question :2
3+3?1>5 2>6 3>2 4>1
1
Wrong

Please enter '2' to next question :2
The end.
Test is 1. True answer 0 of 2.

```

Рис. 2.2: Client.



The image shows two overlapping command prompt windows. The top window, titled 'C:\Windows\system32\cmd.exe', shows the execution of 'client.exe 5001'. It displays the client's connection attempt to 127.0.0.1, login 'ann', and a series of test questions and answers, including a 'Wrong' message. The bottom window, titled 'C:\Users\Аня\Documents\Visual Studio 2010\Projects\server\Debug>server.exe 5001', shows the server's output, including 'MainThread create', 'Number of clients: 1. Accepted client id: 0', 'Number of clients: 2. Accepted client id: 1', and 'Need to kill client 1'.

```

C:\Windows\system32\cmd.exe
C:\Users\Аня\Documents\Visual Studio 2010\Projects\client\Debug>client.exe 5001
Connect 127.0.0.1 OK
Please enter '!' to connect :1
ERROR writing to socket: No error
C:\Users\Аня\Documents\Visual Studio 2010\Projects\client\Debug>

C:\Users\Аня\Documents\Visual Studio 2010\Projects\server\Debug>server.exe 5001
MainThread create
Number of clients: 1. Accepted client id: 0
Number of clients: 2. Accepted client id: 1
k1
Need to kill client 1
k0
Need to kill client 0
-

```

Рис. 2.3: Server/client.

Глава 3

Реализация для работы по протоколу UDP

3.1 Прикладной протокол

Размер пакетов 500 байт, выбор размера проанализирован выше в разделе 1.3. Заметим, что данное задание позволяет это сделать, потому что экзаменатор не предполагает передача слишком больших пакетов и размера в 500 байт будет достаточно для корректной работы протокола.

В связи с этим, изменения в реализации протокола UDP по сравнению с протоколом TCP будут незначительны, и общей архитектуры они не заденут. Подробности в разделе 2.1.

3.2 Архитектура приложения

Особенности архитектуры соответствуют архитектуре приложения для TCP протокола, раздел 2.2

Различия в механизме многопоточности. Не получилось организовать многопоточность с параллельной работой клиентов.

3.3 Тестирование

См. раздел 2.3 .

Глава 4

Выводы

Анализ выполненных заданий, сравнение удобства/эффективности/количества проблем при программировании TCP/UDP

4.1 Реализация для TCP

Теоритические сведения

TCP — ориентированный на соединение протокол, что означает необходимость «рукопожатия» для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

Особенности протокола TCP

- **Надёжность** — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
- **Упорядоченность** — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
- **Тяжеловесность** — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.

- Поточность — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

Анализ

TCP протокол является надежным протоколом с установлением соединения, в связи с чем для TCP клиента помимо создания сокета, необходимо организовать соединение с помощью функции connect.

TCP сервер должен содержать, как минимум, 2 сокета. Один сокет необходим для фиксирования прихода запроса на соединение. После чего для каждого подключившегося клиента создается отдельный сокет.

Это послужило затруднением при реализации многопоточной работы приложения, так как первоначально необходимо создать главный поток, слушающий определенный порт. Как только к серверу подключается клиент, создается новый сокет для этого клиента, после чего он передается в новую нить, содержащую необходимые действия клиента. Главная нить имеет возможность закрыть сокет любого подключенного клиента.

4.2 Реализация для UDP

Теоретические сведения

UDP — более простой, основанный на сообщениях протокол без установления соединения. Протоколы такого типа не устанавливают выделенного соединения между двумя хостами. Связь достигается путем передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя.

Особенности протокола UDP

- Ненадёжный — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
- Неупорядоченность — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
- Легковесность — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.

- Датаграммы — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.

Анализ

Заметим, что структура протокол UDP более простая, чем TCP протокола. Во-первых, нет необходимости использования функции connect, то есть установления адреса и порта по умолчанию для протокола UDP. Однако тогда параметры удалённой стороны будем указывать или получать при каждом вызове операций записи или чтения с помощью функций `sendto` и `recvfrom`.

Однако это создаёт проблемы при попытке реализации многопоточной работы сервера. Так как на каждого клиента не создается отдельного сокета и все клиенты используют 1 сокет, не получилось организовать параллельной работы клиентов. Я использовала мьютексы для распределения доступа к сокету, это не дает возможности параллельной работы клиентов, и новый подключившийся клиент может общаться с сервером только после завершения работы предыдущего клиента.

Приложения

Описание среды разработки

Использованные версии ОС:

Windows 7

Debian GNU/Linux 7.6 (wheezy)

Листинги

Общие для TCP/UDP, Windows/Linux заголовочные файлы и файлы исходных кодов

registration.h (сервер)

```
1 #ifndef REGISTRATION_H
2 #define REGISTRATION_H
3 struct Client{
4     char *login;
5     int numberTest;
6     int sizeQuestion;
7     int sizeTrueAnswer;
8     int sizes[4];
9 };
10 void writeSizeClient(struct Client *c, char (&str)[50]);
11 char *writeLastResult(struct Client *c);
12 void newResult(struct Client *c, int number, int testsize, int
    numberTrueAnswer);
13 void writeClient(struct Client *c, char (&str)[50]);
14 void freeClient(struct Client *c);
15 void newUser(char *login, struct Client *c);
16 int power(int x, int n);
17 int toInt(char *buffer);
18 #endif
```

registration.c (cepbep)

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <malloc.h>
5 #include "registration.h"
6 #include "writeStruct.h"
7 void newUser(char *login, struct Client *c) {
8     c->login=login;
9     c->numberTest = c->sizeQuestion = c->sizeTrueAnswer = 0;
10    c->sizes[0] = c->sizes[1] = c->sizes[2] = 1;
11    c->sizes[3] = strlen(login);
12 }
13 void writeSizeClient(struct Client *c, char (&str)[50]) {
14     int i = 0;
15     int state;
16     for (state = 0; state < 4; state++) {
17         c->sizes[state] = 0;
18     }
19     state = 0;
20     while (str[i] != '/') {
21         if (str[i] == '#') {
22             i++;
23             state++;
24         }
25         c->sizes[state]++;
26         i++;
27     }
28     writeClient(c, str);
29 }
30 void writeClient(struct Client *c, char (&str)[50]) {
31     int i = 0;
32     c->login = (char*) malloc((c->sizes[3]+1) * sizeof(char));
33     c->numberTest = c->sizeQuestion = c->sizeTrueAnswer = 0;
34     int state;
35     int j = 0;
36     int add;
37     int pow;
38
39     for (state = 0; state < 3; state++) {
40         pow = c->sizes[state] - 1;
41         if (str[i] == '#')
42             i++;
43         while (str[i] != '#') {
44             add = ((int) str[i] - '0') * power(10, pow);
45             switch (state) {
46                 case 0:
47                     c->numberTest = c->numberTest + add;
```

```

48         break;
49     case 1:
50         c->sizeQuestion = c->sizeQuestion + add;
51         break;
52     case 2:
53         c->sizeTrueAnswer = c->sizeTrueAnswer + add;
54         break;
55     }
56     i++;
57     pow--;
58 }
59 }
60 state++;
61 i++;
62 j = 0;
63 while (str[i] != '/') {
64
65     c->login[j] = str[i];
66     i++;
67     j++;
68 }
69 c->login[j] = 0;
70 }
71 char *writeLastResult(struct Client *c) {
72     char string[50];
73     sprintf(string, "%d%d%d%s/\n", c->numberTest, c->
74         sizeQuestion, c->sizeTrueAnswer, c->login);
75     return string;
76 }
77 void newResult(struct Client *c, int number, int testsize,
78     int numberTrueAnswer) {
79     c->numberTest = number;
80     c->sizeQuestion = testsize;
81     c->sizeTrueAnswer = numberTrueAnswer;
82 }
83 void freeClient(struct Client *c) {
84     free(c->login);
85 }
86 int power(int x, int n) {
87     int i;
88     int a = 1;
89     for (i = 0; i < n; i++) {
90         a = a * x;
91     }
92     return a;
93 }
94 int toInt(char *buffer){
95     int i=strlen(buffer)-2;

```

```

96     int pow=0;
97     int size=0;
98     while(i>=0){
99         size = size + ((int) buffer[i] - '0') * power
100             (10, pow);
101         pow++;i--;
102     }
103     return size;
104 }

```

writeStruct.h (cepBep)

```

1 #ifndef WRITESTRUCT_H
2 #define WRITESTRUCT_H
3
4 int sizeFile(char *str);
5 int readTrueAnswer( char *str);
6
7 #endif

```

writeStruct.c (cepBep)

```

1
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5 #include "writeStruct.h"
6
7 int readTrueAnswer( char *str) {
8     int i,trueAnswer;
9     int state=0;
10     for (i = 0; i < strlen(str); i++) {
11         if (str[i] == '#')
12             state++;
13         if (str[i] == '!') {
14             trueAnswer = state - 1;
15             break;
16         }
17     }
18     return trueAnswer;
19 }
20 int sizeFile(char *str) {
21     FILE *file;
22     int i = 0;
23     char string[50];
24     file = fopen(str, "r");

```



```

25     while (fgets(string, sizeof(string), file))
26         i++;
27     fclose(file);
28     return i;
29 }

```

write.h (клиент)

```

1  #ifndef REGISTRATION_H
2  #define REGISTRATION_H
3  #define N 4
4  struct Line{
5      int number;
6      char *question;
7      char **answer;
8      int sizeNumber;
9      int *sizeAnswer;
10     int sizeQuest;
11 };
12 void writeStruct (struct Line *x, char *str);
13 void writeSize(struct Line *x, char *str);
14 void freeLine(struct Line *x);
15 void writeToClient(struct Line *x, char *str, char *stringOut);
16 int writeListTest(int *listOfTest, char *buffer, char *string)
17 ;
18 int power(int x, int n);
19 int toInt(char *buffer);
20 int sizeStr(struct Line *x);
21 struct Client{
22     char *login;
23     int numberTest;
24     int sizeQuestion;
25     int sizeTrueAnswer;
26     int sizes[4];
27 };
28 void writeSizeClient(struct Client *c, char *str);
29 void writeClient(struct Client *c, char *str);
30 #endif

```

write.c (клиент)

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include "write.h"
5  #include <string.h>

```

```

6 #include <malloc.h>
7 int toInt(char *buffer){
8     int i=strlen(buffer)-2;
9     int pow=0;
10    int size=0;
11    while(i>=0){
12        size = size + ((int) buffer[i] - '0') * power
            (10, pow);
13        pow++;i--;
14    }
15    return size;
16 }
17 int writeListTest(int *listOfTest,char *buffer,char *string){
18     int j=0;
19     int pow=0;
20     int i=strlen(buffer)-1;
21     strcpy(string,"List of test:");
22     char s[3];
23     while (buffer[i] != '/') {
24         listOfTest[j] =0;
25         while(buffer[i] != '#'){
26             listOfTest[j] = listOfTest[j]
27                 + ((int) buffer[i] - '0') * power(10,
28                     pow);
29             i--;pow++;
30         }
31         i--;j++;
32         pow=0;
33     }
34     int size=j;
35     for(i=0;i<size;i++){
36         sprintf(s, "%d", listOfTest[i]);
37         strcat(string, s);
38     }
39     return size;
40 }
41 int power(int x, int n) {
42     int i;
43     int a = 1;
44     for (i = 0; i < n; i++) {
45         a = a * x;
46     }
47     return a;
48 }
49 void writeSizeClient(struct Client *c, char *str) {
50     int i = 0;
51     int state;
52     for (state = 0; state < 4; state++) {
53         c->sizes[state] = 0;

```

```

53     }
54     state = 0;
55     while (str[i] != '/') {
56         if (str[i] == '#') {
57             i++;
58             state++;
59         }
60         c->sizes[state]++;
61         i++;
62     }
63     writeClient(c, str);
64 }
65 void writeClient(struct Client *c, char *str) {
66     int i = 0;
67     c->login = (char*) malloc((c->sizes[3]+1) * sizeof(char));
68     c->numberTest = c->sizeQuestion = c->sizeTrueAnswer = 0;
69     int state;
70     int j = 0;
71     int add;
72     int pow;
73
74     for (state = 0; state < 3; state++) {
75         pow = c->sizes[state] - 1;
76         if (str[i] == '#')
77             i++;
78         while (str[i] != '#') {
79             add = ((int) str[i] - '0') * power(10, pow);
80             switch (state) {
81                 case 0:
82                     c->numberTest = c->numberTest + add;
83                     break;
84                 case 1:
85                     c->sizeQuestion = c->sizeQuestion + add;
86                     break;
87                 case 2:
88                     c->sizeTrueAnswer = c->sizeTrueAnswer + add;
89                     break;
90             }
91             i++;
92             pow--;
93         }
94     }
95     state++;
96     i++;
97     j = 0;
98     while (str[i] != '/') {
99
100         c->login[j] = str[i];
101         i++;

```

```

102     j++;
103 }
104 c->login[j] = 0;
105 }
106 void freeLine(struct Line *x) {
107     int i;
108     free(x->question);
109     free(x->sizeAnswer);
110     for (i = 0; i < N; i++) {
111         free(x->answer[i]);
112     }
113     free(x->answer);
114 }
115 }
116 void writeToClient(struct Line *x, char *str, char *stringOut)
117 {
118     int i, j, k;
119     for(i=0; i<x->sizeQuest; i++)
120         str[i]=x->question[i];
121     j=i;
122     for (k = 0; k < 4; k++) {
123         str[j++] = k+1+'0';
124         str[j++] = ',';
125         for(i=0; i<x->sizeAnswer[k]; i++)
126             str[j++] = x->answer[k][i];
127         str[j++] = '\n';
128     }
129     str[j++] = '\n';
130     for(i=0; str[i] != '\n'; i++)
131         stringOut[i] = str[i];
132     stringOut[i] = 0;
133 }
134 int sizeStr(struct Line *x){
135     int i;
136     int number = x->sizeQuest+13;
137     for (i = 0; i < N; i++)
138         number = number + x->sizeAnswer[i];
139     return number;
140 }
141 void writeStruct(struct Line *x, char *str) {
142     int i;
143     x->question = (char*) malloc(x->sizeQuest * sizeof(char));
144     x->answer = (char**) malloc(N * sizeof(char*));
145     for (i = 0; i < N; i++) {
146         x->answer[i] = (char*) malloc(x->sizeAnswer[i] * sizeof
            (char));
147     }
148     x->number = 0;

```

```

149     int state = 0;
150     int j = 0;
151     i = 0;
152     while (str[i] != '#') {
153         x->number = x->number
154             + ((int) str[i] - '0') * power(10, x->sizeNumber
155                 - 1 - i);
156         i++;
157     }
158     while (str[i] != '/') {
159         if (str[i] == '#') {
160             state++;
161             i++;
162             j = 0;
163         }
164         if (str[i] == '!')
165             i++;
166         switch (state) {
167             case 1:
168                 x->question[j] = str[i];
169                 break;
170             default:
171                 x->answer[state - 2][j] = str[i];
172                 break;
173         }
174         i++;
175         j++;
176     }
177 void writeSize(struct Line *x, char *str) {
178     x->sizeQuest = 0;
179     x->sizeNumber = 0;
180     x->sizeAnswer = (int*) malloc(N * sizeof(int));
181     int i;
182     for (i = 0; i < N; i++) {
183         x->sizeAnswer[i] = 0;
184     }
185     i = 0;
186     int state = 0;
187     while (str[i] != '/') {
188         if (str[i] == '#') {
189             state++;
190             i++;
191         }
192         if (str[i] == '!')
193             i++;
194         switch (state) {
195             case 0:
196                 x->sizeNumber++;

```

```

197         break;
198     case 1:
199         x->sizeQuest++;
200         break;
201     default:
202         x->sizeAnswer[state - 2]++;
203         break;
204     }
205     i++;
206 }
207 writeStruct(x, str);
208 }

```

Основной файл программы main.c Windows TCP (сервер)

```

1  #include <stdio.h>
2  #include <winsock2.h> // Wincosk2.h должен быть раньше
   windows!
3  #include <windows.h>
4  #include <locale.h>
5  #include <malloc.h>
6  #include <string.h>
7  #include "writeStruct.h"
8  #include "registration.h"
9  #include <cstdlib>
10 #include <iostream>
11 #include <ws2tcpip.h>
12 #include <stdlib.h>
13 using namespace std;
14 #pragma comment(lib, "ws2_32.lib")
15
16 #include <conio.h>
17 #include <windows.h>
18 #include <fstream>
19 int numthread; //число клиентов
20 SOCKET AcceptSocket[100];
21
22 struct client
23 {
24     int id;
25     int port;
26     int socket;
27     bool connected; //подключен ли клиент
28     int num_thread; //номер потока клиента
29 } user[100];
30

```

```

31 #define MY_PORT 5001 // Порт, который слушает сервер 666
32 int port;
33 // прототип функции, обслуживающий подключившихся пользовател
   ей
34 DWORD WINAPI toClient(LPVOID client_socket);
35 DWORD WINAPI AcceptThread(LPVOID lpParam);
36
37 // глобальная переменная - количество активных пользователей
38 int nclients = 0;
39
40 int main(int argc, char *argv[])
41 {
42     setlocale(0, "");
43     char buff[512]; // Буфер для различных нужд
44     if (WSAStartup(0x0202, (WSADATA *)&buff[0]))
45     {
46         // Ошибка!
47         printf("Error_WSAStartup_%.d\n", WSAGetLastError());
48         return -1;
49     }
50     SOCKET mysocket;
51     if ((mysocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
52     {
53         // Ошибка!
54         printf("Error_socket_%.d\n", WSAGetLastError());
55         WSACleanup(); // Деинициализация библиотеки Winsock
56         return -1;
57     }
58     if(argc<2)
59     port=MY_PORT;
60     else
61     port=atoi(argv[1]);
62     HANDLE mainThread;
63     DWORD thID;
64     mainThread = CreateThread(NULL, 0, AcceptThread, (LPVOID)
        mysocket, 0, &thID);
65     if (mainThread== NULL) {
66         printf("Error_CreateThread:_.d\n", GetLastError());
67     }
68     printf("MainThread_create\n");
69     while(true){
70         char text[60]; //cmd by server
71         bool kill_client=false; //нужно ли убить заданного клиен
            та
72         bool exit_server=false;
73         int numb;
74         gets_s(text, 59); //считывание команды сервера
75         if(text[0]=='k'){

```

```

76         kill_client=true; //сервер ввел команду отключения
77         клиента
78         numb=text[1]-48;
79     }
80     else {
81         kill_client=false;
82         break;
83     }
84     if (kill_client==true){ //обработка имени клиента, кот
85     орого необходимо отключить
86     for( int m=0; m < 100; m++){
87         if(user[m].id==numb && user[m].connected == true){
88             printf("Need to kill client %d\n", numb)
89             ;
90             //закрываем сокет клиента
91             user[m].connected = false;
92             closesocket(user[m].socket);
93             numthread--;
94             kill_client=false;
95             break;
96         }
97     }
98 }
99 WSACleanup();
100 return 0;
101 }
102
103 DWORD WINAPI AcceptThread(LPVOID lpParam){
104     int id=0;
105     HANDLE hThreadarr[100];
106     SOCKET mysocket =(SOCKET)lpParam;
107     sockaddr_in local_addr;
108     local_addr.sin_family = AF_INET;
109     local_addr.sin_port = htons(port);
110     local_addr.sin_addr.s_addr = 0;
111     if (bind(mysocket, (sockaddr *)&local_addr, sizeof(
112     local_addr)))
113     {
114         // Ошибка
115         printf("Error bind %d\n", WSAGetLastError());
116         closesocket(mysocket); // закрываем сокет!
117         WSACleanup();
118         return -1;
119     }
120
121     if (listen(mysocket, SOMAXCONN))
122     {
123         // Ошибка
124         printf("Error listen %d\n", WSAGetLastError());

```



```

121         closesocket(mysocket);
122         WSACleanup();
123         return -1;
124     }
125     SOCKET client_socket;
126     sockaddr_in client_addr;
127
128     // Шаг 5 - извлекаем сообщение из очереди
129     while(true){
130
131         int client_addr_size = sizeof(client_addr);
132
133         AcceptSocket[numthread] = accept(mysocket, (sockaddr *)&
            client_addr,&client_addr_size);
134         if (AcceptSocket[numthread] == INVALID_SOCKET) {
135             printf("accept failed\n");
136             break;
137         } else{
138             user[numthread].id = id++;
139             user[numthread].port=ntohs( local_addr.sin_port);
140             user[numthread].socket = AcceptSocket[numthread];
141             user[numthread].connected = true;
142             user[numthread].num_thread = numthread;
143             printf("Number of clients: %d. Accepted client id: %d\n", numthread+1, user[numthread].id);
144         }
145         DWORD thID;
146         hThreadarr[numthread] = CreateThread(NULL, 0, toClient,
            (LPVOID)user[numthread].socket, 0, &thID);
147         if (hThreadarr[numthread]== NULL) {
148             printf("Error CreateThread: %d\n", GetLastError());
149             continue;
150         }
151         numthread++;
152     }
153     for( int i=0; i<numthread; i++){//по завершению закрываем
        все сокеты
154         printf("Close socket %d\n", AcceptSocket[i]);
155         closesocket(AcceptSocket[i]);
156     }
157     DWORD dwEvent;
158     dwEvent = WaitForMultipleObjects( numthread, hThreadarr ,
        TRUE, INFINITE);
159     printf("Threads closed\n");
160     printf("Main socket closed successful\n");
161     return 0;
162 }
163 DWORD WINAPI toClient(LPVOID client_socket)
164 {

```

```

165     SOCKET sock;
166     sock = (SOCKET )client_socket;
167
168     int bytes_recv;
169     char buffer[512];
170     int n=1;
171     char str[50];
172     char result[50];
173     FILE *file;
174     int i, numberTrueAnswer = 0;
175     int numberTest = 0;
176     char *name = (char*) malloc(50 * sizeof(char));
177     while (1) {
178         memset(buffer, 0, sizeof(buffer));
179         n = recv(sock, buffer, 1, 0);
180         n = send(sock, buffer, 1, 0);
181         if (buffer[0] == '!')
182             break;
183     }
184     //Registration
185     int numberClient;
186     int clientSize = sizeFile("registration.txt");
187     struct Client c[50];
188     file = fopen("registration.txt", "r");
189     if (file == NULL) {
190         perror("ERROR_□open_□file");
191         exit(1);
192     }
193     char str1[50];
194     for (i = 0; fgets(str, sizeof(str), file); i++) {
195         strcpy(str1, str);
196         writeSizeClient(&c[i], str1);
197     }
198     fclose(file);
199     char bufferNew[256];
200     numberClient = -1;
201     memset(buffer, 0, sizeof(buffer));
202     memset(bufferNew, 0, sizeof(bufferNew));
203     n = recv(sock, buffer, 255, 0);
204     for (i = 0; i < strlen(buffer) - 1; i++)
205         bufferNew[i] = buffer[i];
206     for (i = 0; i < clientSize; i++) {
207         if (strcmp(bufferNew, c[i].login) == NULL) {
208             numberClient = i;
209             break;
210         }
211     }
212     //New client
213     if (numberClient == -1) {

```

```

214         clientSize++;
215         struct Client client;
216         newUser(bufferNew, &client);
217         c[clientSize - 1] = client;
218         numberClient = clientSize - 1;
219     }
220     strcpy(result, writeLastResult(&c[numberClient]));
221     n = send(sock, result, strlen(result), 0);
222     //List of test
223     while (1) {
224         buffer[0]=0;
225         n = recv(sock, buffer, 1, 0);
226         n = send(sock, buffer, 1, 0);
227         if (buffer[0] == '1')
228             break;
229     }
230     char res[60]="/" ;
231     char s[3];
232     for (i = 50; i >0; i--) {
233         sprintf(name, "%d%s", i, ".txt");
234         if ((file = fopen(name, "r")) != NULL) {
235             sprintf(s, "%d", i);
236             strcat(res, s);
237             fclose(file);
238         }
239     }
240     n = send(sock, res, strlen(res), 0);
241
242     //Number test
243     memset(buffer, 0, sizeof(buffer));
244     n = recv(sock, buffer, 255, 0);
245     numberTest=toInt(buffer);
246     sprintf(name, "%d%s", numberTest, ".txt");
247     file = fopen(name, "r");
248     int testSize = sizeFile(name);
249     int trueAnswer;
250     file = fopen(name, "r");
251     if (file == NULL) {
252         perror("ERROR_□open_□file");
253         exit(1);
254     }
255     int end=0;
256     while(1){
257         if(!fgets(str, sizeof(str), file))
258             end=1;
259         while (1) {
260             memset(buffer, 0, sizeof(buffer));
261             n = recv(sock, buffer,

```

```

262         if(end)
263             buffer[0]='/';
264         n = send(sock, buffer,
265                 1,0);
266         if (buffer[0] == '2' ||
267             buffer[0] == '/')
268             break;
269     }
270     if(end)
271         break;
272     trueAnswer=readTrueAnswer(str);
273     n = send(sock, str, strlen(str),0);
274     //Answer
275     memset(buffer, 0, sizeof(buffer));
276     n = recv(sock, buffer, 1,0);
277     printf("Answer:_%s\n", buffer);
278     if (buffer[0] == trueAnswer+'0') {
279         n = send(sock, "Right\n", 6,0);
280         numberTrueAnswer = numberTrueAnswer + 1;
281     } else
282         n = send(sock, "Wrong\n", 6,0);
283 }
284 sprintf(name, "%d%d%d%s/", numberTest,testSize,
285         numberTrueAnswer,c[numberClient].login);
286 n = send(sock, name, strlen(name),0);
287
288 newResult(&c[numberClient], numberTest, testSize,
289         numberTrueAnswer);
290 file = fopen("registration.txt", "w");
291 if (file == NULL) {
292     perror("ERROR_open_file");
293     exit(1);
294 }
295 for (i = 0; i < clientSize; i++) {
296     fprintf(file, "%d%d%d%s/\n", c[i].numberTest,
297         c[i].sizeQuestion,
298         c[i].sizeTrueAnswer, c[i].login);
299 }
300 fclose(file);
301 free(name);
302 return 0;
303 }

```

Основной файл программы main.c Windows TCP (клиент)

```

1 #include <stdio.h>

```

```

2  #include <winsock2.h> // Winsock2.h должен быть раньше
   windows!
3  #include <windows.h>
4  #include <locale.h>
5  #include <malloc.h>
6  #include <string.h>
7  #include <cstdlib>
8  #include <iostream>
9  #include "write.h"
10
11 #include <ws2tcpip.h>
12 #include <stdlib.h>
13 #pragma comment(lib, "ws2_32.lib")
14 #define PORT 666
15 #define SERVERADDR "127.0.0.1"
16
17 int main(int argc, char* argv[])
18 {
19     char buffer[512];
20
21     if (WSAStartup(0x202, (WSADATA *)&buffer[0]))
22     {
23         printf("WSAStart_ error_ %d\n", WSAGetLastError());
24         return -1;
25     }
26     SOCKET my_sock;
27     my_sock = socket(AF_INET, SOCK_STREAM, 0);
28     if (my_sock < 0)
29     {
30         printf("Socket()_ error_ %d\n", WSAGetLastError());
31         return -1;
32     }
33     sockaddr_in dest_addr;
34     dest_addr.sin_family = AF_INET;
35     dest_addr.sin_port = htons(atoi(argv[1]));
36     HOSTENT *hst;
37     char addr[20];
38     if(argc<3)
39         strcpy(addr, SERVERADDR);
40     else
41         strcpy(addr, argv[2]);
42     if (inet_addr(addr) != INADDR_NONE)
43         dest_addr.sin_addr.s_addr = inet_addr(addr);
44     else
45     {
46         if (hst = gethostbyname(addr))
47             ((unsigned long *)&dest_addr.sin_addr)[0] =
48             ((unsigned long **)hst->h_addr_list)[0][0];
49         else

```

```

50         {
51             printf("Invalid_address_s\n", addr);
52             closesocket(my_sock);
53             WSACleanup();
54             return -1;
55         }
56     }
57     if (connect(my_sock, (sockaddr *)&dest_addr, sizeof(
        dest_addr)))
58     {
59         printf("Connect_error_d\n", WSAGetLastError());
60         return -1;
61     }
62
63     printf("Connect_s_OK\n", addr);
64
65     int n;
66     buffer[0] = '-';
67     while (buffer[0] != '!') {
68         printf("Please_enter_!_to_connect:");
69         memset(buffer, 0, sizeof(buffer));
70         fgets(buffer, 512, stdin);
71         n = send(my_sock, buffer, 1, 0);
72         if (n < 0) {
73             perror("ERROR_writing_to_socket");
74             closesocket(my_sock);
75             WSACleanup();
76             exit(1);
77         }
78         memset(buffer, 0, sizeof(buffer));
79         n = recv(my_sock, buffer, 1, 0);
80         if (n < 0) {
81             perror("ERROR_reading_from_socket");
82             closesocket(my_sock);
83             WSACleanup();
84             exit(1);
85         }
86     }
87     //Registration
88     printf("Enter_login:");
89     memset(buffer, 0, sizeof(buffer));
90     fgets(buffer, 512, stdin);
91     n = send(my_sock, buffer, strlen(buffer), 0);
92     if (n < 0) {
93         perror("ERROR_writing_to_socket");
94         exit(1);
95     }
96     memset(buffer, 0, sizeof(buffer));
97     n = recv(my_sock, buffer, 512, 0);

```

```

98     if (n < 0) {
99         perror("ERROR_reading_from_socket");
100         closesocket(my_sock);
101         WSACleanup();
102         exit(1);
103     }
104     struct Client c;
105     writeSizeClient(&c, buffer);
106     if(c.numberTest==0)
107         printf("Welcome, %s.\n", c.login);
108     else
109         printf("%s your last test is %d. Result %d of %d.\n", c.
110             login, c.numberTest, c.sizeTrueAnswer, c.sizeQuestion);
111     free(c.login);
112     buffer[0] = '-';
113     while (buffer[0] != '1') {
114         printf("Please enter '1' to receive list of test:");
115         memset(buffer, 0, sizeof(buffer));
116         fgets(buffer, 512, stdin);
117         n = send(my_sock, buffer, 1, 0);
118         if (n < 0) {
119             perror("ERROR_writing_to_socket");
120             closesocket(my_sock);
121             WSACleanup();
122             exit(1);
123         }
124         memset(buffer, 0, sizeof(buffer));
125         n = recv(my_sock, buffer, 1, 0);
126         if (n < 0) {
127             perror("ERROR_reading_from_socket");
128             closesocket(my_sock);
129             WSACleanup();
130             exit(1);
131         }
132     }
133     memset(buffer, 0, sizeof(buffer));
134     n = recv(my_sock, buffer, 512, 0);
135     if (n < 0) {
136         perror("ERROR_reading_from_socket");
137         closesocket(my_sock);
138         WSACleanup();
139         exit(1);
140     }
141     int listOfTest[50];
142     int size, number;
143     int trueis=0;
144     char res[50];
145     int i;
146     size=writeListTest(listOfTest, buffer, res);

```

```

146 printf("%s\n", res);
147 while(trueis!=1){
148     printf("Choose test:");
149     memset(buffer, 0, sizeof(buffer));
150     fgets(buffer, 512, stdin);
151     number=toInt(buffer);
152     for(i=0;i<size;i++){
153         if(number==listOfTest[i]){
154             trueis=1;
155             break;
156         }}
157 }
158 n = send(my_sock, buffer, strlen(buffer),0);
159 if (n < 0) {
160     perror("ERROR writing to socket");
161     closesocket(my_sock);
162     WSACleanup();
163     exit(1);
164 }
165 struct Line x;
166 //LOOP
167 while (1) {
168     buffer[0] = '-';
169     while (buffer[0] != '2') {
170         printf("Please enter '2' to next question:");
171         memset(buffer, 0, sizeof(buffer));
172         fgets(buffer, 512, stdin);
173         n = send(my_sock, buffer, 1,0);
174         if (n < 0) {
175             perror("ERROR writing to socket");
176             closesocket(my_sock);
177             WSACleanup();
178             exit(1);
179         }
180         memset(buffer, 0, sizeof(buffer));
181         n = recv(my_sock, buffer, 1,0);
182         if (n < 0) {
183             perror("ERROR reading from socket");
184             closesocket(my_sock);
185             WSACleanup();
186             exit(1);
187         }
188         if(buffer[0]=='/')
189             break;
190     }
191     if(buffer[0]=='/')
192         break;
193     memset(buffer, 0, sizeof(buffer));
194     n = recv(my_sock, buffer,512,0);

```



```

195         if (n < 0) {
196             perror("ERROR_reading_from_socket");
197             closesocket(my_sock);
198             WSACleanup();
199             exit(1);
200         }
201         writeSize(&x,buffer);
202         int numb=sizeStr(&x);
203         char str[50];
204         char* stringOut = (char*) malloc((numb) * sizeof(
                char));
205         writeToClient(&x,str,stringOut);
206         printf("%s\n", stringOut);
207
208         freeLine(&x);
209         while(1){
210             memset(buffer, 0, sizeof(buffer));
211             fgets(buffer, 512, stdin);
212             if(!strcmp(buffer,"1\n") || !strcmp(buffer,"2\n") ||
                !strcmp(buffer,"3\n") || !strcmp(buffer,"4\n") )
213                 break;
214             else
215                 printf("Enter_your_answer(1,2,3,4):\n");
216         }
217         n = send(my_sock, buffer, 1,0);
218         if (n < 0) {
219             perror("ERROR_writing_to_socket");
220             closesocket(my_sock);
221             WSACleanup();
222             exit(1);
223         }
224         memset(buffer, 0, sizeof(buffer));
225         n = recv(my_sock, buffer, 6,0);
226         if (n < 0) {
227             perror("ERROR_reading_from_socket");
228             closesocket(my_sock);
229             WSACleanup();
230             exit(1);
231         }
232         printf("%s\n",buffer);
233     }
234     n = recv(my_sock, buffer, 512,0);
235     if (n < 0) {
236         perror("ERROR_reading_from_socket");
237         closesocket(my_sock);
238         WSACleanup();
239         exit(1);
240     }
241     struct Client c1;

```

```

242 writeSizeClient(&c1, buffer);
243 printf("The_end.\nTest_is_%d.True_answer_%d_of_%d.\n", c1.
    numberTest, c1.sizeTrueAnswer, c1.sizeQuestion );
244 free(c1.login);
245     closesocket(my_sock);
246     WSACleanup();
247     exit(1);
248 }

```

Основной файл программы main.c Linux TCP (сервер)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <string.h>
7 #include "writeStruct.h"
8 #include "registration.h"
9 #include <time.h>
10 #include <netdb.h>
11 #include <pthread.h>
12 pthread_t t1;
13
14 void* thread1(int sock){
15     printf("New_client\n");
16     char buffer[512];
17     int n;
18     char str[50];
19     char result[50];
20     FILE *file;
21     int i, numberTrueAnswer = 0;
22     char *clientFile = "/home/user/workspace/server/
    registration.txt";
23     int numberTest = 0;
24     char *name = (char*) malloc(50 * sizeof(char));
25     while (1) {
26         bzero(buffer, 512);
27         n = read(sock, buffer, 1);
28         if (n < 0) {
29             perror("ERROR_reading_from_socket");
30             exit(1);
31         }
32         n = write(sock, buffer, 1);
33         if (n < 0) {
34             perror("ERROR_writing_to_socket");
35             exit(1);

```

```

36     }
37     if (buffer[0] == '!')
38         break;
39 }
40 //Registration
41 int numberClient;
42 int clientSize = sizeFile(clientFile);
43 struct Client c[50];
44 file = fopen(clientFile, "r");
45 if (file == NULL) {
46     perror("ERROR_open_file");
47     exit(1);
48 }
49 for (i = 0; fgets(str, sizeof(str), file); i++) {
50     writeSizeClient(&c[i], &str);
51 }
52 fclose(file);
53 char bufferNew[512];
54 numberClient = -1;
55 bzero(buffer, 512);
56 bzero(bufferNew, 512);
57 n = read(sock, buffer, 512);
58 if (n < 0) {
59     perror("ERROR_reading_from_socket");
60     exit(1);
61 }
62 for (i = 0; i < strlen(buffer) - 1; i++)
63     bufferNew[i] = buffer[i];
64 for (i = 0; i < clientSize; i++) {
65     if (strcmp(bufferNew, c[i].login) == NULL) {
66         numberClient = i;
67         break;
68     }
69 }
70 //New client
71 if (numberClient == -1) {
72     clientSize++;
73     struct Client client;
74     newUser(bufferNew, &client);
75     c[clientSize - 1] = client;
76     numberClient = clientSize - 1;
77 }
78 char strres[50];
79 sprintf(strres, "%d#%d#%d#s/\n", c[numberClient].
    numberTest,
80     c[numberClient].sizeQuestion, c[numberClient].
    sizeTrueAnswer, c[numberClient].login);
81 strcpy(result, strres);
82 n = write(sock, result, strlen(result));

```

```

83         if (n < 0) {
84             perror("ERROR_writing_to_socket");
85             exit(1);
86         }
87         //List of test
88         while (1) {
89             bzero(buffer, 512);
90             n = read(sock, buffer, 1);
91             if (n < 0) {
92                 perror("ERROR_reading_from_socket");
93                 exit(1);
94             }
95             n = write(sock, buffer, 1);
96             if (n < 0) {
97                 perror("ERROR_writing_to_socket");
98                 exit(1);
99             }
100             if (buffer[0] == '1')
101                 break;
102         }
103         char res[60]="/" ;
104         char s[3];
105         for (i = 50; i > 0; i--) {
106             sprintf(name, "%s%d%s", "/home/user/workspace/
server/test/", i, ".txt");
107             if ((file = fopen(name, "r")) != NULL) {
108                 sprintf(s, "%d", i);
109                 strcat(res, s);
110                 fclose(file);
111             }
112         }
113         n = write(sock, res, strlen(res));
114         if (n < 0) {
115             perror("ERROR_writing_to_socket");
116             exit(1);
117         }
118
119         //Number test
120         bzero(buffer, 512);
121         n = read(sock, buffer, 512);
122         if (n < 0) {
123             perror("ERROR_reading_from_socket");
124             exit(1);
125         }
126         numberTest=toInt(buffer);
127         sprintf(name, "%s%d%s", "/home/user/workspace/
server/test/", numberTest,
128             ".txt");

```

```

129         file = fopen(name, "r");
130     int testSize = sizeFile(name);
131     int trueAnswer;
132     file = fopen(name, "r");
133     if (file == NULL) {
134         perror("ERROR_open_file");
135         exit(1);
136     }
137     int end=0;
138     while(1){
139         if(!fgets(str, sizeof(str), file))
140             end=1;
141         while (1) {
142             bzero(buffer, 512);
143             n = read(sock, buffer, 1)
144                 ;
145             if (n < 0) {
146                 perror("ERROR_reading_
147                     from_socket");
148                 exit(1);
149             }
150             if(end)
151                 buffer[0]='/';
152             n = write(sock, buffer,
153                 1);
154             if (n < 0) {
155                 perror("ERROR_writing_
156                     to_socket");
157                 exit(1);
158             }
159             if (buffer[0] == '2' ||
160                 buffer[0] == '/')
161                 break;
162         }
163     if(end)
164         break;
165     trueAnswer=readTrueAnswer(str);
166     n = write(sock, str, strlen(str));
167     if (n < 0) {
168         perror("ERROR_writing_to_socket");
169         exit(1);
170     }
171     //Answer
172     bzero(buffer, 512);
173     n = read(sock, buffer, 1);
174     if (n < 0) {
175         perror("ERROR_reading_from_socket");
176         exit(1);
177     }

```

```

173         printf("Answer:_%s\n", buffer);
174         if (buffer[0] == trueAnswer+'0') {
175             n = write(sock, "Right\n", 6);
176             if (n < 0) {
177                 perror("ERROR_writing_to_socket");
178                 exit(1);
179             }
180             numberTrueAnswer = numberTrueAnswer + 1;
181         } else
182             n = write(sock, "Wrong\n", 6);
183         if (n < 0) {
184             perror("ERROR_writing_to_socket");
185             exit(1);
186         }
187     }
188     sprintf(name, "%d#%d#%d#s/", numberTest, testSize,
        numberTrueAnswer, c[numberClient].login);
189     n = write(sock, name, strlen(name));
190     if (n < 0) {
191         perror("ERROR_writing_to_socket");
192         exit(1);
193     }
194
195
196     newResult(&c[numberClient], numberTest, testSize,
        numberTrueAnswer);
197     file = fopen("/home/user/workspace/server/
        registration.txt", "w");
198     if (file == NULL) {
199         perror("ERROR_open_file");
200         exit(1);
201     }
202     for (i = 0; i < clientSize; i++) {
203         fprintf(file, "%d#%d#%d#s/\n", c[i].numberTest,
            c[i].sizeQuestion,
204             c[i].sizeTrueAnswer, c[i].login);
205     };
206     fclose(file);
207     free(name);
208 }
209 int main(int argc, char *argv[]) {
210
211     int sockfd, newsockfd, portno, clilen;
212     const int on = 1;
213     char buffer[512];
214     struct sockaddr_in serv_addr, cli_addr;
215     int n;
216
217     sockfd = socket(AF_INET, SOCK_STREAM, 0);

```

```

218     if (sockfd < 0) {
219         perror("ERROR_□opening_□socket");
220         exit(1);
221     }
222     //setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof
        (on));
223     bzero((char *) &serv_addr, sizeof(serv_addr));
224     //portno = 5001;
225     if (argc < 2) {
226         printf("usage_□default_□port_□5001\n");
227         portno = 5001;
228     }
229     else
230     portno = atoi(argv[1]);
231     serv_addr.sin_family = AF_INET;
232     serv_addr.sin_addr.s_addr = INADDR_ANY;
233     serv_addr.sin_port = htons(portno);
234
235     if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(
        serv_addr)) < 0) {
236         perror("ERROR_□on_□binding");
237         exit(1);
238     }
239     listen(sockfd, 5);
240     clilen = sizeof(cli_addr);
241     bzero(buffer, 512);
242     while(1){
243         newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
            &clilen);
244         if (newsockfd < 0) {
245             perror("ERROR_□on_□accept");
246             exit(1);
247         }
248         pthread_create(&t1, NULL, thread1, newsockfd);
249     }
250     pthread_join(&t1, NULL);
251     return 0;
252 }

```

Основной файл программы main.c Linux TCP (кли- ент)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>

```

```

6 #include <string.h>
7 #include "write.h"
8
9 int main(int argc, char *argv[]) {
10     int sockfd, portno, n;
11     struct sockaddr_in serv_addr;
12     struct hostent *server;
13     char end;
14     char buffer[512];
15     int i;
16
17     if (argc < 3) {
18         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
19         exit(0);
20     }
21     portno = atoi(argv[2]);
22     /* Create a socket point */
23     sockfd = socket(AF_INET, SOCK_STREAM, 0);
24     if (sockfd < 0) {
25         perror("ERROR_opening_socket");
26         exit(1);
27     }
28     server = gethostbyname(argv[1]);
29     if (server == NULL) {
30         fprintf(stderr, "ERROR_no_such_host\n");
31         exit(0);
32     }
33
34     bzero((char *) &serv_addr, sizeof(serv_addr));
35     serv_addr.sin_family = AF_INET;
36     /* bcopy((char *)server->h_addr,
37      (char *)&serv_addr.sin_addr.s_addr,
38      server->h_length);*/
39     serv_addr.sin_port = htons(portno);
40
41     /* Now connect to the server */
42     if (connect(sockfd, &serv_addr, sizeof(serv_addr)) < 0) {
43         perror("ERROR_connecting");
44         exit(1);
45     }
46     /* Now ask for a message from the user, this message
47      * will be read by server
48      */
49     //Start connect
50     buffer[0] = '-';
51     while (buffer[0] != '!') {
52         printf("Please_enter_!'_to_connect:");
53         bzero(buffer, 512);
54         fgets(buffer, 512, stdin);

```



```

55     n = write(sockfd, buffer, 1);
56     if (n < 0) {
57         perror("ERROR_writing_to_socket");
58         exit(1);
59     }
60     bzero(buffer, 512);
61     n = read(sockfd, buffer, 1);
62     if (n < 0) {
63         perror("ERROR_reading_from_socket");
64         exit(1);
65     }
66 }
67 //Registration
68 printf("Enter_login:");
69 bzero(buffer, 512);
70 fgets(buffer, 512, stdin);
71 n = write(sockfd, buffer, strlen(buffer));
72 if (n < 0) {
73     perror("ERROR_writing_to_socket");
74     exit(1);
75 }
76 bzero(buffer, 512);
77 n = read(sockfd, buffer, 512);
78 if (n < 0) {
79     perror("ERROR_reading_from_socket");
80     exit(1);
81 }
82 struct Client c;
83 writeSizeClient(&c, buffer);
84 if(c.numberTest==0)
85     printf("Welcome, %s.\n", c.login);
86 else
87     printf("%s_your_last_test_is_%d. Result_%d_of_%d.\n", c.
88         login, c.numberTest, c.sizeTrueAnswer, c.sizeQuestion);
89 free(c.login);
90 buffer[0] = '-';
91 while (buffer[0] != '1') {
92     printf("Please_enter '1' to receive list of test:");
93     bzero(buffer, 512);
94     fgets(buffer, 512, stdin);
95     n = write(sockfd, buffer, 1);
96     if (n < 0) {
97         perror("ERROR_writing_to_socket");
98         exit(1);
99     }
100     bzero(buffer, 512);
101     n = read(sockfd, buffer, 1);
102     if (n < 0) {
103         perror("ERROR_reading_from_socket");

```

```

103         exit(1);
104     }
105 }
106 bzero(buffer, 512);
107 n = read(sockfd, buffer, 512);
108     if (n < 0) {
109         perror("ERROR_reading_from_socket");
110         exit(1);
111     }
112 int listOfTest[50];
113 int size, number;
114 int true=0;
115 char res[50];
116 size=writeListTest(listOfTest,buffer,res);
117 printf("%s\n", res);
118 while(true!=1){
119     printf("Choose_test:");
120     bzero(buffer, 512);
121     fgets(buffer, 512, stdin);
122     number=toInt(buffer);
123     for(i=0;i<size;i++){
124         if(number==listOfTest[i]){
125             true=1;
126             break;
127         }}
128 }
129 n = write(sockfd, buffer, strlen(buffer));
130     if (n < 0) {
131         perror("ERROR_writing_to_socket");
132         exit(1);
133     }
134 struct Line x;
135 //LOOP
136 while (1) {
137     buffer[0] = '-';
138     while (buffer[0] != '2') {
139         printf("Please_enter_'2'_to_next_question:");
140         bzero(buffer, 512);
141         fgets(buffer, 512, stdin);
142         n = write(sockfd, buffer, 1);
143         if (n < 0) {
144             perror("ERROR_writing_to_socket");
145             exit(1);
146         }
147         bzero(buffer, 512);
148         n = read(sockfd, buffer, 1);
149         if (n < 0) {
150             perror("ERROR_reading_from_socket");
151             exit(1);

```

```

152         }
153         if(buffer[0]=='/')
154             break;
155     }
156     if(buffer[0]=='/')
157         break;
158     bzero(buffer, 512);
159     n = read(sockfd, buffer, 512);
160     if (n < 0) {
161         perror("ERROR_reading_from_socket");
162         exit(1);
163     }
164     writeSize(&x,buffer);
165     char str[sizeStr(&x)];
166     char* stringOut = (char*) malloc(50 * sizeof(char));
167     writeToClient(&x,str,stringOut);
168     printf("%s\n", stringOut);
169
170     freeLine(&x);
171     while(1){
172         bzero(buffer, 512);
173         fgets(buffer, 512, stdin);
174         if(!strcmp(buffer,"1\n") || !strcmp(buffer,"2\n") ||
175            !strcmp(buffer,"3\n") || !strcmp(buffer,"4\n") )
176             break;
177         else
178             printf("Enter your answer (1,2,3,4):\n");
179     }
180     n = write(sockfd, buffer, 1);
181     if (n < 0) {
182         perror("ERROR_writing_to_socket");
183         exit(1);
184     }
185     bzero(buffer, 512);
186     n = read(sockfd, buffer, 6);
187     if (n < 0) {
188         perror("ERROR_reading_from_socket");
189         exit(1);
190     }
191     n = read(sockfd, buffer, 512);
192     if (n < 0) {
193         perror("ERROR_reading_from_socket");
194         exit(1);
195     }
196     struct Client c1;
197     writeSizeClient(&c1, &buffer);
198     printf("The end.\nTest is %d. True answer %d of %d.\n", c1.
        numberTest, c1.sizeTrueAnswer, c1.sizeQuestion);

```

```

199 free(c1.login);
200             close(sockfd);
201     return 0;
202 }

```

Основной файл программы main.c Windows UDP (сервер)

```

1  #include <stdio.h>
2  #include <winsock2.h> // Winsock2.h должен быть раньше
   windows!
3  #include <windows.h>
4  #include <locale.h>
5  #include <malloc.h>
6  #include <string.h>
7  #include "writeStruct.h"
8  #include "registration.h"
9  #include <cstdlib>
10 #include <iostream>
11 #include <stdlib.h>
12
13 #pragma comment(lib,"ws2_32.lib") //Winsock Library
14
15 #define BUFLen 500 //Max length of buffer
16 #define PORT 5001 //The port on which to listen for
   incoming data
17
18 int main(int argc, char *argv[])
19 {
20     SOCKET sock;
21     struct sockaddr_in server, si_other;
22     int slen , recv_len;
23     char buf[BUFLen];
24     WSADATA wsa;
25
26     slen = sizeof(si_other) ;
27     if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
28     {
29         printf("Failed. Error Code: %d",WSAGetLastError());
30         exit(EXIT_FAILURE);
31     }
32     if((sock = socket(AF_INET , SOCK_DGRAM , 0 )) ==
        INVALID_SOCKET)
33     {
34         printf("Could not create socket: %d" ,
            WSAGetLastError());
35     }

```

```

36 int portno;
37     if (argc < 2) {
38         printf("usage_default_port_5001\n");
39         portno = PORT;
40     }
41     else
42     portno = atoi(argv[1]);
43     server.sin_family = AF_INET;
44     server.sin_addr.s_addr = INADDR_ANY;
45     server.sin_port = htons( portno);
46
47     //Bind
48     if( bind(sock ,(struct sockaddr *)&server , sizeof(server
49         )) == SOCKET_ERROR)
50     {
51         printf("Bind_failed_with_error_code_:_%d" ,
52             WSAGetLastError());
53         exit(EXIT_FAILURE);
54     }
55     while(1){
56         char buffer[BUFLen];
57         int n=1;
58         char str[50];
59         char result[50];
60         FILE *file;
61         int i, numberTrueAnswer = 0;
62         int numberTest = 0;
63         char *name = (char*) malloc(50 * sizeof(char));
64         while (1) {
65             memset(buffer, 0, sizeof(buffer));
66             n = recvfrom(sock, buffer, 1, 0,(struct sockaddr
67                 *) &si_other, &slen);
68             n = sendto(sock, buffer, 1, 0,(struct sockaddr*)
69                 &si_other, slen) ;
70             if (buffer[0] == '!')
71                 break;
72         }
73         //Registration
74         int numberClient;
75         int clientSize = sizeFile("registration.txt");
76         struct Client c[50];
77         file = fopen("registration.txt", "r");
78         if (file == NULL) {
79             perror("ERROR_open_file");
80             exit(1);
81         }
82         char str1[50];
83         for (i = 0; fgets(str, sizeof(str), file); i++) {
84             strcpy(str1,str);

```

```

81         writeSizeClient(&c[i], str1);
82     }
83     fclose(file);
84     char bufferNew[256];
85     numberClient = -1;
86     memset(buffer, 0, sizeof(buffer));
87     memset(bufferNew, 0, sizeof(bufferNew));
88     n = recvfrom(sock, buffer, BUFLen, 0, (struct
        sockaddr *) &si_other, &slen);
89     for (i = 0; i < strlen(buffer) - 1; i++)
90         bufferNew[i] = buffer[i];
91     for (i = 0; i < clientSize; i++) {
92         if (strcmp(bufferNew, c[i].login) == NULL) {
93             numberClient = i;
94             break;
95         }
96     }
97     //New client
98     if (numberClient == -1) {
99         clientSize++;
100         struct Client client;
101         newUser(bufferNew, &client);
102         c[clientSize - 1] = client;
103         numberClient = clientSize - 1;
104     }
105     strcpy(result, writeLastResult(&c[numberClient]));
106     n = sendto(sock, result, strlen(result), 0, (struct
        sockaddr*) &si_other, slen) ;
107     //List of test
108     while (1) {
109         buffer[0]=0;
110         n = recvfrom(sock, buffer, 1, 0, (struct sockaddr
            *) &si_other, &slen) ;
111         n = sendto(sock, buffer, 1, 0, (struct sockaddr*)
            &si_other, slen) ;
112         if (buffer[0] == '1')
113             break;
114     }
115     char res[60]="/" ;
116     char s[3];
117     for (i = 50; i >0; i--) {
118         sprintf(name, "%d%s", i, ".txt");
119         if ((file = fopen(name, "r")) != NULL) {
120             sprintf(s, "%d", i);
121             strcat(res, s);
122             fclose(file);
123         }
124     }

```

```

125         n = sendto(sock, res, strlen(res), 0, (struct
            sockaddr*) &si_other, slen) ;
126
127     //Number test
128     memset(buffer, 0, sizeof(buffer));
129     n = recvfrom(sock, buffer, BUFLen, 0, (struct
        sockaddr*) &si_other, &slen) ;
130     numberTest=toInt(buffer);
131     sprintf(name, "%d%s", numberTest, ".txt");
132     file = fopen(name, "r");
133     int testSize = sizeFile(name);
134     int trueAnswer;
135     file = fopen(name, "r");
136     if (file == NULL) {
137         perror("ERROR_␣open_␣file");
138         exit(1);
139     }
140     int end=0;
141     while(1){
142         if(!fgets(str, sizeof(str), file))
143             end=1;
144         while (1) {
145             memset(buffer, 0, sizeof(buffer));
146             n = recvfrom(sock, buffer
                , 1, 0, (struct
                    sockaddr *) &si_other,
                    &slen);
147             if(end)
148                 buffer[0]='\0';
149             n = sendto(sock, buffer,
                1, 0, (struct sockaddr*)
                    &si_other, slen) ;
150             if (buffer[0] == '2' ||
                buffer[0] == '/')
151                 break;
152         }
153         if(end)
154             break;
155         trueAnswer=readTrueAnswer(str);
156         n = sendto(sock, str, strlen(str), 0, (struct
            sockaddr*) &si_other, slen) ;
157         //Answer
158         memset(buffer, 0, sizeof(buffer));
159         n = recvfrom(sock, buffer, 1, 0, (struct sockaddr
            *) &si_other, &slen);
160         printf("Answer:␣%s\n", buffer);
161         if (buffer[0] == trueAnswer+'0') {
162             n = sendto(sock, "Right\n", 6, 0, (struct
                sockaddr*) &si_other, slen) ;

```

```

163         numberTrueAnswer = numberTrueAnswer + 1;
164     } else
165     {
166         n = sendto(sock, "Wrong\n", 6, 0, (struct
167             sockaddr*) &si_other, slen);
168     }
169     sprintf(name, "%d#%d#%d#s/", numberTest, testSize,
170         numberTrueAnswer, c[numberClient].login);
171     n = sendto(sock, name, strlen(name), 0, (struct
172         sockaddr*) &si_other, slen);
173
174     newResult(&c[numberClient], numberTest, testSize,
175         numberTrueAnswer);
176     file = fopen("registration.txt", "w");
177     if (file == NULL) {
178         perror("ERROR_□open_□file");
179         exit(1);
180     }
181     for (i = 0; i < clientSize; i++) {
182         fprintf(file, "%d#%d#%d#s/\n", c[i].numberTest,
183             c[i].sizeQuestion,
184             c[i].sizeTrueAnswer, c[i].login);
185     };
186     fclose(file);
187     free(name);
188     return 0;
189 }
190
191 // closesocket(s);
192 // WSACleanup();

```

Основной файл программы main.c Windows UDP (кли- ент)

```

1 #include <stdio.h>
2 #include <winsock2.h> // Wincosk2.h должен быть раньше
   windows!
3 #include <windows.h>
4 #include <locale.h>
5 #include <malloc.h>
6 #include <string.h>
7 #include <cstdlib>
8 #include <iostream>
9 #include "write.h"
10
11 #pragma comment(lib, "ws2_32.lib") //Winsock Library

```



```

12
13 #define SERVER "127.0.0.1" //ip address of udp server
14 #define BUFLLEN 500 //Max length of buffer
15
16 int main(int argc, char**argv)
17 {
18     struct sockaddr_in si_other;
19     int my_sock, slen=sizeof(si_other);
20     char buffer[BUFLLEN];
21     char message[BUFLLEN];
22     WSADATA wsa;
23     if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
24     {
25         printf("Failed. Error Code: %d",WSAGetLastError());
26         exit(EXIT_FAILURE);
27     }
28     if ( (my_sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))
29         == SOCKET_ERROR)
30     {
31         printf("socket() failed with error code: %d" ,
32             WSAGetLastError());
33         exit(EXIT_FAILURE);
34     }
35     memset((char *) &si_other, 0, sizeof(si_other));
36     si_other.sin_family = AF_INET;
37     char addr[20];
38     if(argc<3)
39         strcpy(addr,SERVER);
40     else
41         strcpy(addr,argv[2]);
42     si_other.sin_port = htons(atoi(argv[1]));
43     si_other.sin_addr.S_un.S_addr = inet_addr(addr);
44
45     int n;
46     buffer[0] = '-';
47     while (buffer[0] != '!') {
48         printf("Please enter '!' to connect:");
49         memset(buffer, 0, sizeof(buffer));
50         fgets(buffer, BUFLLEN, stdin);
51         n = sendto(my_sock, buffer, 1,0, (struct sockaddr *) &
52             si_other, slen);
53         if (n < 0) {
54             perror("ERROR writing to socket");
55             exit(1);
56         }
57         memset(buffer, 0, sizeof(buffer));
58         n=recvfrom(my_sock, buffer, 1, 0, (struct sockaddr *) &
59             si_other, &slen);
60         if (n < 0) {

```

```

57         perror("ERROR_reading_from_socket");
58         exit(1);
59     }
60 }
61 printf("Enter_login:");
62 memset(buffer, 0, sizeof(buffer));
63 fgets(buffer, BUFLen, stdin);
64 n = sendto(my_sock, buffer, strlen(buffer), 0, (struct
        sockaddr *) &si_other, slen);
65 if (n < 0) {
66     perror("ERROR_writing_to_socket");
67     exit(1);
68 }
69 memset(buffer, 0, sizeof(buffer));
70 n = recvfrom(my_sock, buffer, BUFLen, 0, (struct sockaddr
        *) &si_other, &slen);
71 if (n < 0) {
72     perror("ERROR_reading_from_socket");
73     exit(1);
74 }
75 struct Client c;
76 writeSizeClient(&c, buffer);
77 if(c.numberTest==0)
78     printf("Welcome, %s.\n", c.login);
79 else
80     printf("%s your last test is %d. Result %d of %d.\n", c.
        login, c.numberTest, c.sizeTrueAnswer, c.sizeQuestion);
81 free(c.login);
82 buffer[0] = '-';
83 while (buffer[0] != '1') {
84     printf("Please enter '1' to receive list of test:");
85     memset(buffer, 0, sizeof(buffer));
86     fgets(buffer, BUFLen, stdin);
87     n = sendto(my_sock, buffer, 1, 0, (struct sockaddr *) &
        si_other, slen);
88     if (n < 0) {
89         perror("ERROR_writing_to_socket");
90         exit(1);
91     }
92     memset(buffer, 0, sizeof(buffer));
93     n = recvfrom(my_sock, buffer, 1, 0, (struct sockaddr *)
        &si_other, &slen);
94     if (n < 0) {
95         perror("ERROR_reading_from_socket");
96         exit(1);
97     }
98 }
99 memset(buffer, 0, sizeof(buffer));

```

```

100     n = recvfrom(my_sock, buffer, BUFLen, 0, (struct sockaddr
        *) &si_other, &slen);
101         if (n < 0) {
102             perror("ERROR_reading_from_socket");
103             exit(1);
104         }
105     int listOfTest[50];
106     int size, number;
107     int trueis=0;
108     char res[50];
109     int i;
110     size=writeListTest(listOfTest,buffer,res);
111     printf("%s\n", res);
112     while(trueis!=1){
113         printf("Choose_test:_");
114         memset(buffer, 0, sizeof(buffer));
115         fgets(buffer, BUFLen, stdin);
116         number=toInt(buffer);
117         for(i=0;i<size;i++){
118             if(number==listOfTest[i]){
119                 trueis=1;
120                 break;
121             }}
122     }
123     n = sendto(my_sock, buffer, strlen(buffer), 0, (struct
        sockaddr *) &si_other, slen);
124         if (n < 0) {
125             perror("ERROR_writing_to_socket");
126             exit(1);
127         }
128     struct Line x;
129     //LOOP
130     while (1) {
131         buffer[0] = '-';
132         while (buffer[0] != '2') {
133             printf("Please_enter_'2'_to_next_question:");
134             memset(buffer, 0, sizeof(buffer));
135             fgets(buffer, BUFLen, stdin);
136             n = sendto(my_sock, buffer, 1, 0, (struct sockaddr
                *) &si_other, slen);
137             if (n < 0) {
138                 perror("ERROR_writing_to_socket");
139                 exit(1);
140             }
141             memset(buffer, 0, sizeof(buffer));
142             n = recvfrom(my_sock, buffer, 1, 0, (struct
                sockaddr *) &si_other, &slen);
143             if (n < 0) {
144                 perror("ERROR_reading_from_socket");

```

```

145         exit(1);
146     }
147     if(buffer[0]=='/')
148         break;
149 }
150 if(buffer[0]=='/')
151     break;
152     memset(buffer, 0, sizeof(buffer));
153     n = recvfrom(my_sock, buffer, BUFLen,0, (
        struct sockaddr *) &si_other, &slen);
154     if (n < 0) {
155         perror("ERROR_reading_from_socket");
156         exit(1);
157     }
158     writeSize(&x,buffer);
159     int numb=sizeStr(&x);
160     char str[50];
161     char* stringOut = (char*) malloc((numb) * sizeof(
        char));
162     writeToClient(&x,str,stringOut);
163     printf("%s\n", stringOut);
164
165     freeLine(&x);
166     while(1){
167         memset(buffer, 0, sizeof(buffer));
168         fgets(buffer, BUFLen, stdin);
169         if(!strcmp(buffer,"1\n") || !strcmp(buffer,"2\n") ||
            !strcmp(buffer,"3\n") || !strcmp(buffer,"4\n") )
170             break;
171         else
172             printf("Enter_your_answer(1,2,3,4):\n");
173     }
174     n = sendto(my_sock, buffer, 1,0, (struct sockaddr *) &
        si_other, slen);
175     if (n < 0) {
176         perror("ERROR_writing_to_socket");
177         exit(1);
178     }
179     memset(buffer, 0, sizeof(buffer));
180     n = recvfrom(my_sock, buffer, 6,0, (struct sockaddr *)
        &si_other, &slen);
181     if (n < 0) {
182         perror("ERROR_reading_from_socket");
183         exit(1);
184     }
185     printf("%s\n",buffer);
186 }
187 n = recvfrom(my_sock, buffer, BUFLen,0, (struct sockaddr
    *) &si_other, &slen);

```

```

188         if (n < 0) {
189             perror("ERROR_reading_from_socket");
190             exit(1);
191         }
192         struct Client c1;
193         writeSizeClient(&c1, buffer);
194         printf("The_end.\nTest_is_%d.True_answer_%d_of_%d.\n", c1.
            numberTest, c1.sizeTrueAnswer, c1.sizeQuestion );
195         free(c1.login);
196         closesocket(my_sock);
197         WSACleanup();
198
199         return 0;
200     }

```

Основной файл программы main.c Linux UDP (сервер)

```

1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <stdio.h>
4  #include <netdb.h>
5  #include <pthread.h>
6  #include "writeStruct.h"
7  #include "registration.h"
8  #define maxSize 500
9  pthread_t t1;
10 pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;;
11 static void* worker(void* arg) {
12     pthread_mutex_lock(&mutex);
13     int sock = *(int*)arg;
14     printf("Worker_for_%d_is_up\n", sock);
15     int n;
16     char buffer[maxSize];
17     while(1){
18         struct sockaddr_in cliaddr;
19         socklen_t len = sizeof(struct sockaddr_in);
20         char str[50];
21         char result[50];
22         FILE *file;
23         struct Client c[50];
24         int i, numberTrueAnswer = 0;
25         char *clientFile = "/home/user/workspace/udp_server/
            registration.txt";
26         char numberTest = '0';
27         char *name = (char*) malloc(50 * sizeof(char));
28
29         while (1) {

```

```

30         bzero(buffer, maxSize);
31         n = recvfrom(sock, buffer, 1,0,(struct
            sockaddr *) &cliaddr, &len);
32         if (n < 0) {
33             perror("ERROR_reading_from_socket");
34             exit(1);
35         }
36         n = sendto(sock, buffer, 1,0,(struct sockaddr
            *) &cliaddr, len);
37         if (n < 0) {
38             perror("ERROR_writing_to_socket");
39             exit(1);
40         }
41         if (buffer[0] == '!')
42             break;
43     }
44     //Registration
45     int numberClient;
46     int clientSize = sizeFile(clientFile);
47     file = fopen(clientFile, "r");
48     if (file == NULL) {
49         perror("ERROR_open_file");
50         exit(1);
51     }
52     for (i = 0; fgets(str, sizeof(str), file); i++) {
53         writeSizeClient(&c[i], &str);
54     }
55     fclose(file);
56     char bufferNew[maxSize];
57     numberClient = -1;
58     bzero(buffer, maxSize);
59     bzero(bufferNew, maxSize);
60     n = recvfrom(sock, buffer, maxSize,0,(struct
            sockaddr *) &cliaddr, &len);
61     if (n < 0) {
62         perror("ERROR_reading_from_socket");
63         exit(1);
64     }
65     for (i = 0; i < strlen(buffer) - 1; i++)
66         bufferNew[i] = buffer[i];
67     for (i = 0; i < clientSize; i++) {
68         if (strcmp(bufferNew, c[i].login) == NULL) {
69             numberClient = i;
70             break;
71         }
72     }
73     //New client
74     if (numberClient == -1) {
75         clientSize++;

```

```

76         struct Client client;
77         new(bufferNew, &client);
78         c[clientSize - 1] = client;
79         numberClient = clientSize - 1;
80     }
81     char stres[50];
82     sprintf(stres, "%d%d%d%s/\n", c[
83         numberClient].numberTest,
84         c[numberClient].sizeQuestion, c[
85         numberClient].sizeTrueAnswer, c[
86         numberClient].login);
87     strcpy(result, stres);
88     n = sendto(sock, result, strlen(result), 0, (struct
89     sockaddr *) &cliaddr, len);
90     if (n < 0) {
91         perror("ERROR_writing_to_socket");
92         exit(1);
93     }
94     //List of test
95     while (1) {
96         bzero(buffer, maxSize);
97         n = recvfrom(sock, buffer, 1, 0, (
98         struct sockaddr *) &cliaddr, &
99         len);
100         if (n < 0) {
101             perror("ERROR_reading_from_
102             socket");
103             exit(1);
104         }
105         n = sendto(sock, buffer, 1, 0, (
106         struct sockaddr *) &cliaddr,
107         len);
108         if (n < 0) {
109             perror("ERROR_writing_to_socket
110             ");
111             exit(1);
112         }
113         if (buffer[0] == '1')
114             break;
115     }
116     char res[60] = "/";
117     char s[3];
118     for (i = 50; i > 0; i--) {
119         sprintf(name, "%s%d%s", "/home/user/workspace/
120         udp_server/test/", i, ".txt");
121         if ((file = fopen(name, "r")) != NULL) {
122             sprintf(s, "%d", i);
123             strcat(res, s);
124             fclose(file);

```

```

114     }
115 }
116 n = sendto(sock, res, strlen(res), 0, (struct
    sockaddr *) &cliaddr, len);
117 if (n < 0) {
118     perror("ERROR_writing_to_socket");
119     exit(1);
120 }
121
122 //Number test
123 bzero(buffer, maxSize);
124 n = recvfrom(sock, buffer, maxSize, 0, (struct
    sockaddr *) &cliaddr, &len);
125 if (n < 0) {
126     perror("ERROR_reading_from_socket");
127     exit(1);
128 }
129 numberTest = toInt(buffer);
130 sprintf(name, "%s%d%s", "/home/user/workspace/
    udp_server/test/", numberTest,
    ".txt");
131 file = fopen(name, "r");
132 int testSize = sizeFile(name);
133 int trueAnswer;
134 file = fopen(name, "r");
135 if (file == NULL) {
136     perror("ERROR_open_file");
137     exit(1);
138 }
139
140 int end = 0;
141 while(1){
142     if(!fgets(str, sizeof(str), file))
143         end = 1;
144     while (1) {
145         bzero(buffer, maxSize);
146         n = recvfrom(sock,
            buffer, 1, 0, (struct
            sockaddr *) &
            cliaddr, &len);
147         if (n < 0) {
148             perror("ERROR_
                reading_from_
                socket");
149             exit(1);
150         }
151         if(end)
152             buffer[0] = '/';
153         n = sendto(sock,
            buffer, 1, 0, (struct

```



```

154         sockaddr *) &
155         cliaddr, len);
156         if (n < 0) {
157             perror("ERROR_writing_to_socket");
158             exit(1);
159         }
160         if (buffer[0] == '2'
161             || buffer[0] == '/')
162             break;
163     }
164     if(end)
165         break;
166     trueAnswer=readTrueAnswer(str);
167     n = sendto(sock, str, strlen(str),0,(struct
168         sockaddr *) &cliaddr, len);
169     if (n < 0) {
170         perror("ERROR_writing_to_socket");
171         exit(1);
172     }
173     //Answer
174     bzero(buffer, maxSize);
175     n = recvfrom(sock, buffer, 1,0,(struct
176         sockaddr *) &cliaddr, &len);
177     if (n < 0) {
178         perror("ERROR_reading_from_socket");
179         exit(1);
180     }
181     printf("Answer: %s\n", buffer);
182     if (buffer[0] == trueAnswer+'0') {
183         n = sendto(sock, "Right\n", 6,0,(struct
184             sockaddr *) &cliaddr, len);
185         if (n < 0) {
186             perror("ERROR_writing_to_socket");
187             exit(1);
188         }
189         numberTrueAnswer = numberTrueAnswer + 1;
190     } else
191         n = sendto(sock, "Wrong\n", 6,0,(struct
192             sockaddr *) &cliaddr, len);
193     if (n < 0) {
194         perror("ERROR_writing_to_socket");
195         exit(1);
196     }
197 }
198 sprintf(name, "%d#%d#%d#s/", numberTest,testSize
199     , numberTrueAnswer,c[numberClient].login);

```

```

192         n = sendto(sock, name, strlen(name), 0, (struct
            sockaddr *) &cliaddr, len);
193     if (n < 0) {
194         perror("ERROR_writing_to_socket");
195         exit(1);
196     }
197
198
199     newResult(&c[numberClient], numberTest, testSize,
        numberTrueAnswer);
200     file = fopen("/home/user/workspace/server/
        registration.txt", "w");
201     if (file == NULL) {
202         perror("ERROR_open_file");
203         exit(1);
204     }
205     for (i = 0; i < clientSize; i++) {
206         fprintf(file, "%d%d%d%s\n", c[i].
            numberTest, c[i].sizeQuestion,
            c[i].sizeTrueAnswer, c[i].login);
207     };
208     fclose(file);
209     free(name);
210 }
211
212
213 pthread_mutex_unlock(&mutex);
214 shutdown(sock, 2);
215 close(sock);
216 }
217
218 int main(int argc, char**argv) {
219     int sock;
220     int portno=5001;
221     //pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
222     if(argc<2)
223 printf("use_default_port_5001\n");
224     else portno=atoi(argv[1]);
225     struct sockaddr_in servaddr, cliaddr;
226     pthread_t accept_thread;
227     sock = socket(AF_INET, SOCK_DGRAM, 0);
228
229     bzero(&servaddr, sizeof(servaddr));
230     servaddr.sin_family = AF_INET;
231     servaddr.sin_addr.s_addr = INADDR_ANY;
232     servaddr.sin_port = htons(portno);
233     int optval = 1;
234     setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &optval,
        sizeof optval);

```

```

235     bind(sock, (struct sockaddr *) &servaddr, sizeof(servaddr)
        );
236 listen(sock,5);
237     pthread_create(&(accept_thread),NULL, worker, (void*)
        &sock);
238     while (1) {
239         char command = getchar();
240         if (command == 'q') {
241             break;
242             //int pthread_mutex_destroy(pthread_mutex_t *
                mutex);
243         }
244     }
245     printf("Exit...\n");
246     shutdown(sock, 2);
247     close(sock);
248     pthread_join(accept_thread, NULL);
249     printf("Done\n");
250     return 0;
251 }

```

Основной файл программы main.c Linux UDP (кли- ент)

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <netinet/in.h>
6  #include <string.h>
7  #include "write.h"
8  #include <pthread.h>
9  #include <strings.h>
10 #include <string.h>
11 #include <errno.h>
12 #include <assert.h>
13 #define maxSize 500
14 int main(int argc, char**argv)
15 {
16
17     int sockfd,n;
18     struct sockaddr_in servaddr,cliaddr;
19     char buffer[maxSize];
20     if (argc < 3) {
21         fprintf(stderr, "usage_%s_hostname_port\n", argv[0]);
22         exit(0);
23     }

```

```

24
25 sockfd=socket(AF_INET,SOCK_DGRAM,0);
26
27 bzero(&servaddr,sizeof(servaddr));
28 servaddr.sin_family = AF_INET;
29 servaddr.sin_addr.s_addr=inet_addr(argv[1]);
30 servaddr.sin_port=htons(atoi(argv[2]));
31 if (connect(sockfd,&servaddr,sizeof(servaddr)) < 0)
32 {
33     perror("ERROR_connecting");
34     exit(1);
35 }
36 //Start connect
37 buffer[0] = '-';
38 while (buffer[0] != '!') {
39     printf("Please_enter_'!'_to_connect:");
40     bzero(buffer, maxSize);
41     fgets(buffer, maxSize, stdin);
42     n = write(sockfd,buffer,1);
43     if (n < 0) {
44         perror("ERROR_writing_to_socket");
45         exit(1);
46     }
47     bzero(buffer, maxSize);
48     n = read(sockfd, buffer, 1);
49     if (n < 0) {
50         perror("ERROR_reading_from_socket");
51         exit(1);
52     }
53 }
54 //Registration
55 printf("Enter_login:");
56 bzero(buffer, maxSize);
57 fgets(buffer, maxSize, stdin);
58 n = write(sockfd, buffer, strlen(buffer));
59 if (n < 0) {
60     perror("ERROR_writing_to_socket");
61     exit(1);
62 }
63 bzero(buffer, maxSize);
64 n = read(sockfd, buffer, maxSize);
65 if (n < 0) {
66     perror("ERROR_reading_from_socket");
67     exit(1);
68 }
69 struct Client c;
70 writeSizeClient(&c, buffer);
71 if(c.numberTest==0)
72     printf("Welcome,_%s.\n",c.login);

```

```

73     else
74     printf("%s your last test is %d. Result %d of %d.\n"
        , c.login, c.numberTest, c.sizeQuestion, c.
        sizeTrueAnswer);
75     free(c.login);
76     buffer[0] = '-';
77     while (buffer[0] != '1') {
78         printf("Please enter '1' to receive list of test
        :");
79         bzero(buffer, maxSize);
80         fgets(buffer, maxSize, stdin);
81         n = write(sockfd, buffer, 1);
82         if (n < 0) {
83             perror("ERROR writing to socket");
84             exit(1);
85         }
86         bzero(buffer, maxSize);
87         n = read(sockfd, buffer, 1);
88         if (n < 0) {
89             perror("ERROR reading from socket");
90             exit(1);
91         }
92     }
93     bzero(buffer, maxSize);
94     n = read(sockfd, buffer, maxSize);
95     if (n < 0) {
96         perror("ERROR reading from socket");
97         exit(1);
98     }
99     int listOfTest[50];
100    int size, number;
101    int true=0;
102    char res[50];
103    size=writeListTest(listOfTest,buffer,res);
104    printf("%s\n", res);
105    while(true!=1){
106        printf("Choose test:");
107        bzero(buffer, maxSize);
108        fgets(buffer, maxSize, stdin);
109        number=toInt(buffer);
110        int i;
111        for(i=0;i<size;i++){
112            if(number==listOfTest[i]){
113                true=1;
114                break;
115            }
116        }
117        n = write(sockfd, buffer, strlen(buffer));
118        if (n < 0) {

```

```

119         perror("ERROR_writing_to_socket");
120         exit(1);
121     }
122     struct Line x;
123     //LOOP
124     while (1) {
125         buffer[0] = '-';
126         while (buffer[0] != '2') {
127             printf("Please_enter_2'_to_next_question_");
128             bzero(buffer, maxSize);
129             fgets(buffer, maxSize, stdin);
130             n = write(sockfd, buffer, 1);
131             if (n < 0) {
132                 perror("ERROR_writing_to_socket");
133                 exit(1);
134             }
135             bzero(buffer, maxSize);
136             n = read(sockfd, buffer, 1);
137             if (n < 0) {
138                 perror("ERROR_reading_from_socket");
139                 exit(1);
140             }
141             if(buffer[0]=='/')
142                 break;
143         }
144         if(buffer[0]=='/')
145             break;
146         bzero(buffer, maxSize);
147         n = read(sockfd, buffer, maxSize);
148         if (n < 0) {
149             perror("ERROR_reading_from_socket");
150             exit(1);
151         }
152         writeSize(&x,buffer);
153         char str[sizeStr(&x)];
154         char* stringOut = (char*) malloc(50 * sizeof(
            char));
155         writeToClient(&x,str,stringOut);
156         printf("%s\n", stringOut);
157
158         freeLine(&x);
159         while(1){
160             bzero(buffer, maxSize);
161             fgets(buffer, maxSize, stdin);
162             if(!strcmp(buffer,"1\n") || !strcmp(buffer,"2\n"
                ) || !strcmp(buffer,"3\n") || !strcmp(buffer
                ,"4\n") )

```

```

163         break;
164     else
165         printf("Enter your answer(1,2,3,4):\n");
166     }
167     n = write(sockfd, buffer, 1);
168     if (n < 0) {
169         perror("ERROR writing to socket");
170         exit(1);
171     }
172     bzero(buffer, maxSize);
173     n = read(sockfd, buffer, 6);
174     if (n < 0) {
175         perror("ERROR reading from socket");
176         exit(1);
177     }
178 }
179 n = read(sockfd, buffer, maxSize);
180 if (n < 0) {
181     perror("ERROR reading from socket");
182     exit(1);
183 }
184 struct Client c1;
185 writeSizeClient(&c1, &buffer);
186 printf("The end.\n Test is %d. True answer %d of %d.\n"
187        , c1.numberTest, c1.sizeTrueAnswer, c1.sizeQuestion);
187 free(c1.login);
188 close(sockfd);
189 return 0;
190 }

```

Файл сборки Makefile TCP, UDP (клиент)

```

1 all:  main.c  write.h write.c
2      gcc main.c write.c -o client

```

Файл сборки Makefile TCP, UDP (сервер)

```

1 all:  main.c writeStruct.c writeStruct.h  registration.c
      registration.h
2      gcc main.c writeStruct.c registration.c -o server -
      lpthread

```