

# Todo list

|  |    |
|--|----|
| ■ Не забыть вставить все исходники . . . . . | 15 |
|--|----|

# Сети ЭВМ и телекоммуникации

А. В. Никитина

25 ноября 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. Когда клиент ответит на все вопросы теста, сервер выдает результат клиенту и символ окончания соединения (например, тот же символ "!"). После получения результата клиент должен разорвать соединение.

### 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 (верный символ '!' при запросах).  
Остальные пакеты могут иметь максимальный размер 256 символов.
- **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<br>1+2+3+login/                  | сервер отвечает клиенту о регистрации в сети       | strlen(str)      |
| TESTS-REQUEST | !                                    | клиент запрашивает список тестов                   | 1                |
| TESTS-ANSWER  | str1<br>1+2+3.../                    | сервер выдает клиенту список тестов                | strlen(str1)     |
| CHOICE-TEST   | numb                                 | клиент отправляет номер теста                      | strlen(numb)     |
| CHOICE-OK     | !                                    | сервер подтверждает получение номера теста         | 1                |
| QUEST-REQUEST | !                                    | клиент запрашивает наличие следующего вопрос       | 1                |
| QUEST-YES     | !                                    | сервер отвечает что следующий вопрос есть          | 1                |
| QUEST-NO      | /                                    | сервер отвечает что следующего вопроса нет         | 1                |
| QUESTION      | quest                                | сервер отправляет вопрос клиенту                   | strlen(question) |
| ANSWER        | answer=<br>(1,2,3,4)                 | клиент отправляет ответ серверу                    | 1                |
| TRUE          | true=<br>(Right, Wrong) <sup>5</sup> | сервер отправляет клиенту правильность ответа      | 6                |
| RESULT        | result<br>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

**Последовательность и форматы пакетов:**

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



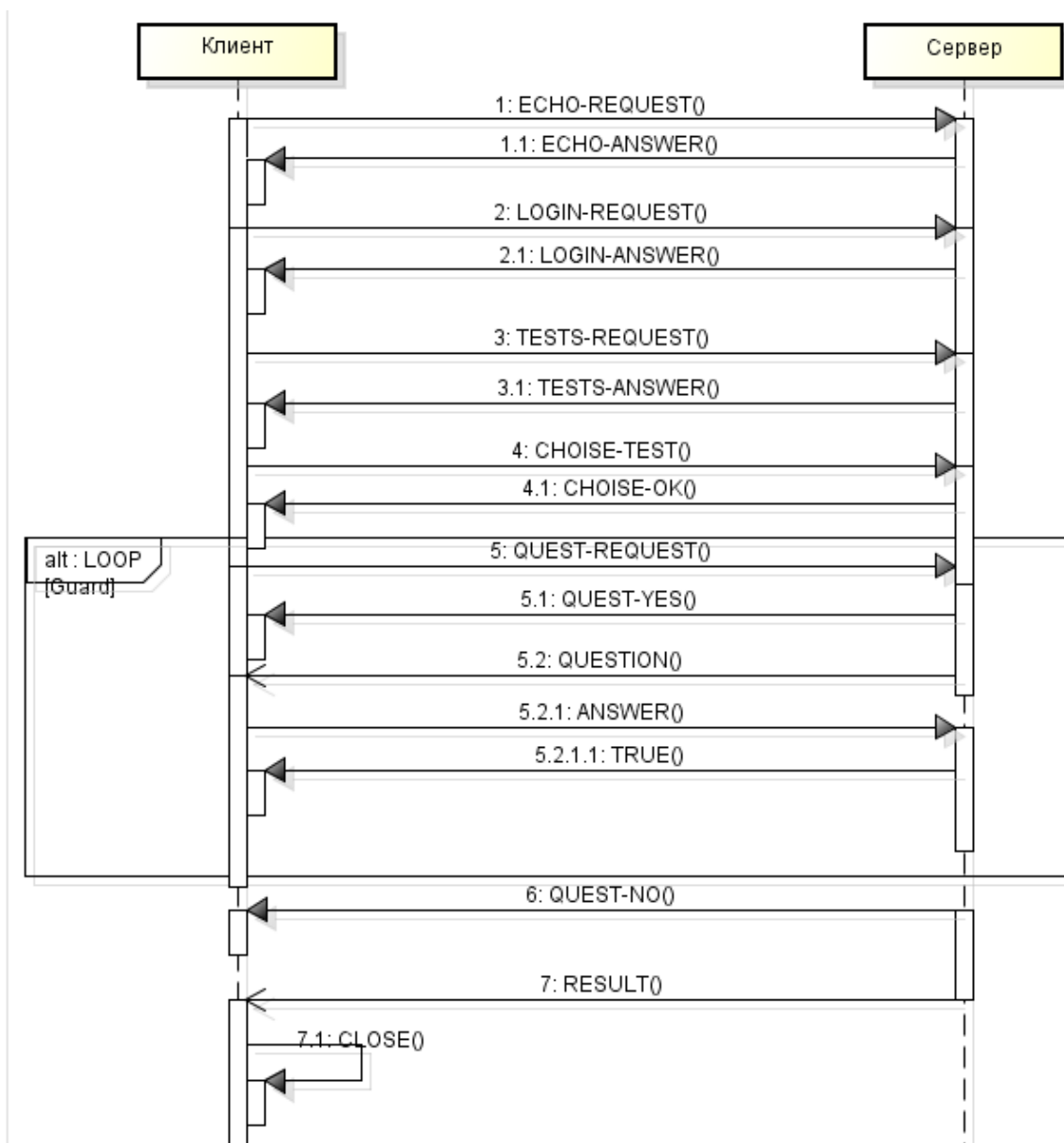


Рис. 2.1: Sequence Diagram.

## 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 '!'.  
!<br>
- После принятия этого символа, сервер разрешает клиенту передачу смысловых сообщений, отправив в ответ также сообщение длиной 1 символ '!'.  
!<br>
- 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 '!')
- 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 '!')
- Начинается цикл прохождения теста:
  1. Клиент отправляет запрос на получение следующего вопроса (строка '! ', длиной 1 символ)
  2. Сервер передает строку, подтверждающую наличие вопроса (строка '! ', 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).

## 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 строки 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 приложений запускается сервер «Удаленного тестирования» и несколько клиентов. В процессе тестирования проверяются основные возможности сервера по параллельному тестированию нескольких пользователей.

### 2.3.2 Тестовый план и результаты тестирования

По шагам, с перечнем входных данных

## Глава 3

# Реализация для работы по протоколу UDP

### 3.1 Прикладной протокол

Например, в табличном виде – набор и формат команд, размеры полей (для создания таблиц можно пользоваться Wizard -> Quick tabular)

В случае незначительных изменений допустимо перечислить их со ссылкой на раздел ??, описывающий протокол для взаимодействия по TCP .

### 3.2 Архитектура приложения

Особенности архитектуры и ограничения (желательно с графической схемой)

### 3.3 Тестирование

#### 3.3.1 Описание тестового стенда и методики тестирования

#### 3.3.2 Тестовый план и результаты тестирования

По шагам, с перечнем входных данных, а также методика тестирования поведения программы на потерю, дублирование и перемешивание дейтаграмм

## Глава 4

### Выводы

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

#### 4.1 Реализация для TCP

#### 4.2 Реализация для UDP



# Приложения

## Описание среды разработки

Версии ОС, компиляторов, утилит, и проч., которые использовались в процессе разработки

## Листинги

Основной файл программы `main.c`

Файл сборки `Makefile`

Не забыть вставить все исходники