

# Todo list

■ Не забыть вставить все исходники . . . . .	11
--	----

# Сети ЭВМ и телекоммуникации

А. В. Никитина

7 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, пока 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 соединения, если это не сделано за фиксированное время (пусть будет 2 минуты), сервер должен принудительно отключить клиента.

### **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 номера выбранного текста формируем пакеты длиной=1, так как больше нам и не требуется.

## **1.3 Накладываемые ограничения**

## Глава 2

# Реализация для работы по протоколу TSP

### 2.1 Прикладной протокол

#### Последовательность и форматы пакетов:

После запуска сервер начинает прослушивать определенный порт, он ждет подключение от клиента. Клиент может подключиться к серверу, отправив сообщение "!".

После удачного соединения. На стороне клиента появляется сообщение

Please enter "yes" to registration:

При вводе клиентом "yes сервер просит ввести новый логин и проверяет его с уже существующими логинами, хранящимися в файле registration.txt, при совпадении просит ввести логин вновь. При вводе нового логина, сервер просит ввести пароль, после чего сохраняет новую строку в файле registration.txt в виде (формат данного файла описан ниже)

0 0 0 0 login parol/

При вводе клиентом любой строки, кроме "yes сервер просит ввести уже существующий логин и проверяет его с существующими логинами. При совпадении, сервер просит ввести пароль и проверяет его с паролем данного пользователя, если 3 раза введен неверный пароль, выводится опять сообщение о вводе логина.

При идентификации клиента, сервер отправляет ему результат последнего тестирования в виде строки

login, your last test is 1. True answers is 12 of 13 answers.

Далее идет цикл выборки теста, сервер просит ввести название теста да тех пор, пока клиент не введет название имеющегося в папке с тестами. При верном значении начинается последовательная передача пакетов от сервера клиенту, содержащих вопрос и варианты ответов.

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

Клиент должен отправить цифру от 1 до 4, указывающую на выбранный ответ. В ответ сервер отправляет "right" либо "wrong" в зависимости от правильности ответа. 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 модифицирует файл `registration.txt`, записывая в строку соответствующего пользователя номер пройденного теста, количество вопросов теста и верных ответов клиента.

Далее сервер выдает клиенту его результат, например такой.

Number of question 15  
Number of true answer 7!

Последний символ '!', обозначает конец соединения. Получив этот символ, клиент обрывает соединение функцией `close(s)`.

## 2.2 Архитектура приложения

### Формат исходных файлов.

Каждый тест хранится в отдельном текстовом файле с именем, соответствующим номеру теста (`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`

$1 \mp 2 \mp 3 \mp login \mp parol /$

' $\mp$ '-знак-разделитель, разделяет элементы строки

'/'-знак конца строки

Элементы строки в порядке их следования:

- номер теста
- количество вопросов теста

- количество верных ответов
- логин пользователя
- пароль 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
- `char *parol`-пароль пользователя
- `int sizes[5]`- 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 входе в систему под именем зарегистрированного пользователя, сервер отправляет клиенту строку с результатом последнего тестирования с помощью функции

```
char *writeLastResult(struct Client *c);
```

При регистрации нового пользователя последовательно вызываются функции

```
void new(char *login,char *parol,struct Client *c);
void registration(struct Client *c,int reWrite);
```

Первая функция создает нового пользователя, у которого первые 3 элемента =0. Вторая функция записывает в файл `registration.txt` строку



`0 ≠ 0 ≠ 0 ≠ login ≠ parol/`

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 ≠ Howareyou? ≠ 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 строки 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);
```

Для отправки вопроса клиенту, следует обратно собрать его в строку для наглядного предоставления информации

```
char *writeToClient(struct Line *x);
```

## 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