

1 Создание сервера обмена сообщениями по протоколу TCP

1.1 Принцип работы в сетях TCP/IP

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Протокол *TCP* — протокол транспортного уровня согласно модели *OSI/ISO* (см. рис. ??).

Основное достоинство протокола – ориентированность на “качество” соединения. *TCP* устанавливает предварительное соединение, а затем проверяет, действительно ли файл дошёл до получателя [kumar_survey_2012].

На рис. 1 показано, как устанавливается соединения между устройствами по протоколу *TCP* (технология тройного рукопожатия):

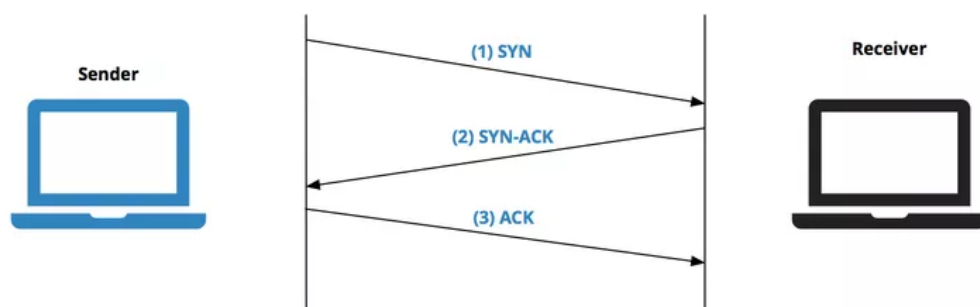


Рис. 1. Установка соединения протоколом *TCP*

1. Клиент отправляет серверу пакет с флагом SYN. Этот начальный пакет позволяет клиенту установить, каким должен быть первый порядковый номер для пакетов запроса;
2. Сервер, получив пакет, добавляет к нему флаг ACK и отправляет клиенту. Этот пакет подтверждает порядковый номер, отправленный клиентом, подтверждая его. Однако сервер также должен отправить SYN и порядко-

вый номер обратно клиенту, чтобы установить, каким должен быть первый порядковый номер для пакетов ответа, исходящих от сервера;

3. Наконец, клиент отвечает на пакет SYN / ACK пакетом ACK, который подтверждает запрос порядкового номера сервера.

1.2 Принцип работы программы

Программа состоит из двух частей:

1. Клиент;
2. Сервер;

Программа работает следующим образом (см. рис. 2):

1. Производится запуск сервера;
2. Производится запуск клиента;
3. С клиента поступает запрос на подключение, который всегда одобряется;
4. После подключения сервер отправляет клиенту сообщение об успешном подключении (этого можно было не делать, поскольку успешность подключения обеспечивается протоколом *TCP*).

После этого клиент может передать серверу файл. Контроль за передачей файла ведётся из консоли сервера, а также в графическом интерфейсе клиента.

1.3 Реализация кода программы

1. **Запуск сервера.** На листинге 1 приведён метод, осуществляющий запуск сервера на порт 9999.

```

1  void myserver::startServer ()
2  {
3      if ( this->listen (QHostAddress::Any ,9999))
4      {
5          qDebug() << "Listening";
6      }
7      else
8      {
9          qDebug() << "Not listening";
10     }
11 }

```

Листинг 1. Метод запуска сервера

2. При поступлении входящего запроса реализуется метод, приведённый в листинге 2:

```

1  void myserver::incomingConnection ( qintptr
    socketDescriptor )
2  {
3      socket = new QTcpSocket ( this );
4      socket->setSocketDescriptor ( socketDescriptor );
5      connect ( socket , SIGNAL ( readyRead () ) , this , SLOT (
        sockReady () ) );
6      connect ( socket , SIGNAL ( disconnected () ) , this , SLOT (
        sockDisc () ) );
7      qDebug() << socketDescriptor << " Client connected";

8      socket->write ( "You are connect" );
9      qDebug() << "Send client connect status - YES";
10 }

```

Листинг 2. Метод входящего подключения

1.4 Терминал сервера

Сервер не нуждается в графической оболочке, ему достаточно выводить текст в консоль посредством встроенных в Qt средств (`qDebug()`). На рис. 3 показан вывод при запуске сервера.

На рис. 4 показан вывод сообщения при подключении клиента к сервера. 5 – уникальный номер соединения.

На последнем рис. 5 выведено сообщение при получении сервером файла.

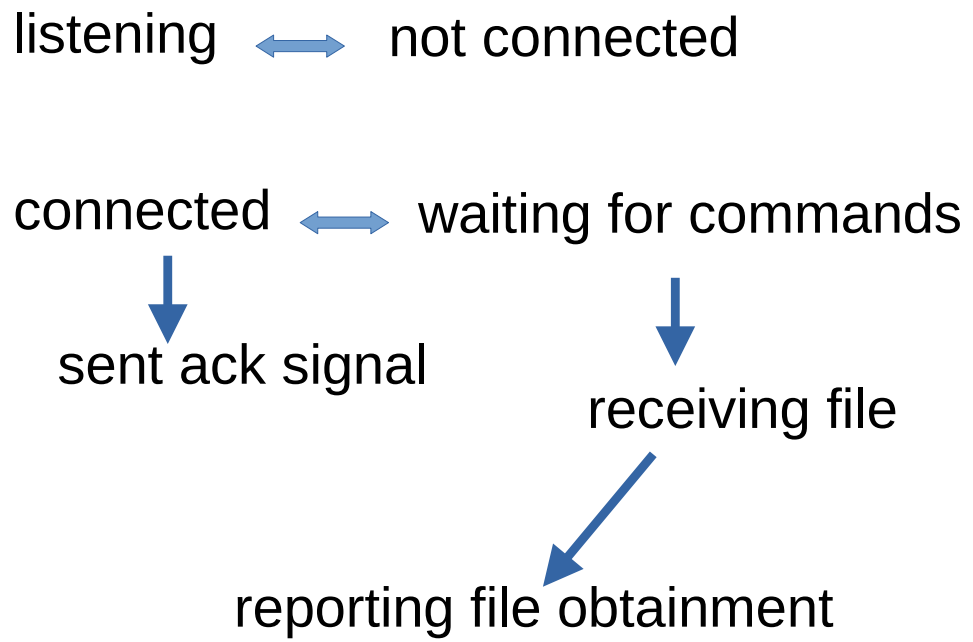


Рис. 2. Схема работы серверной части

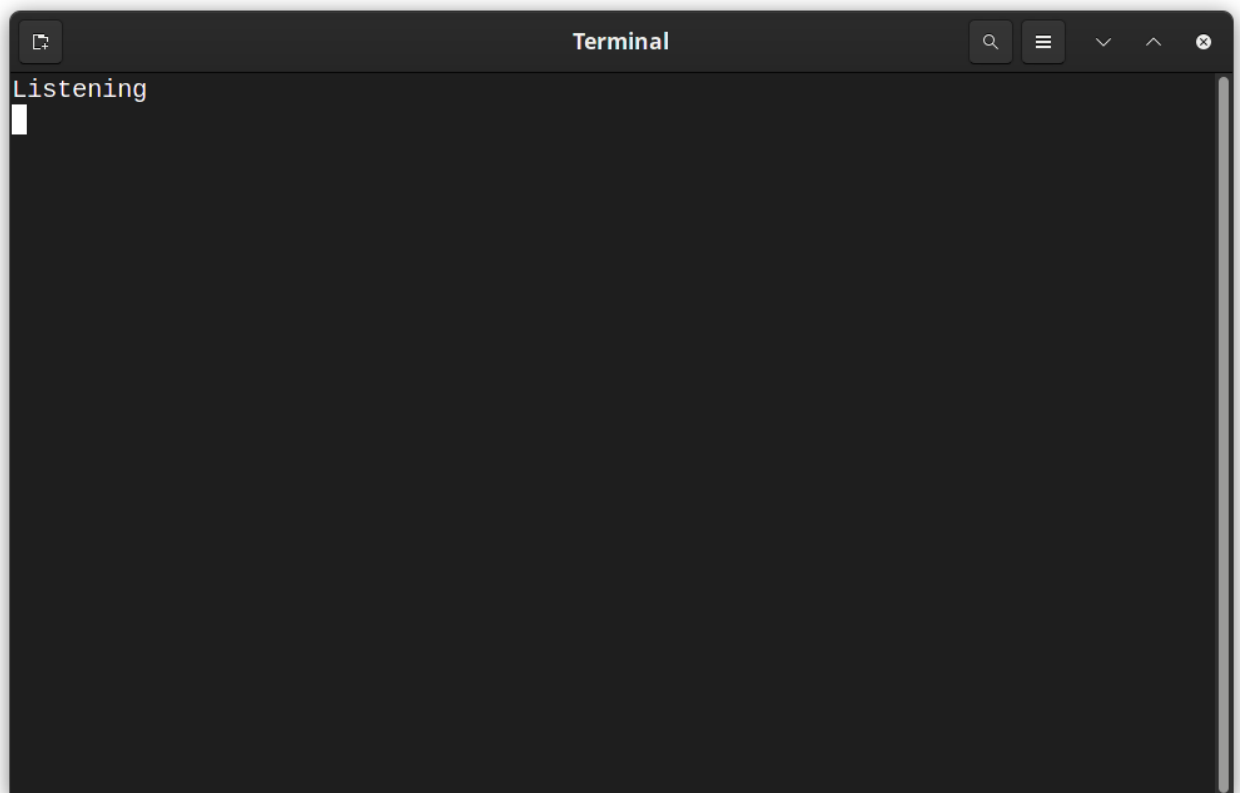
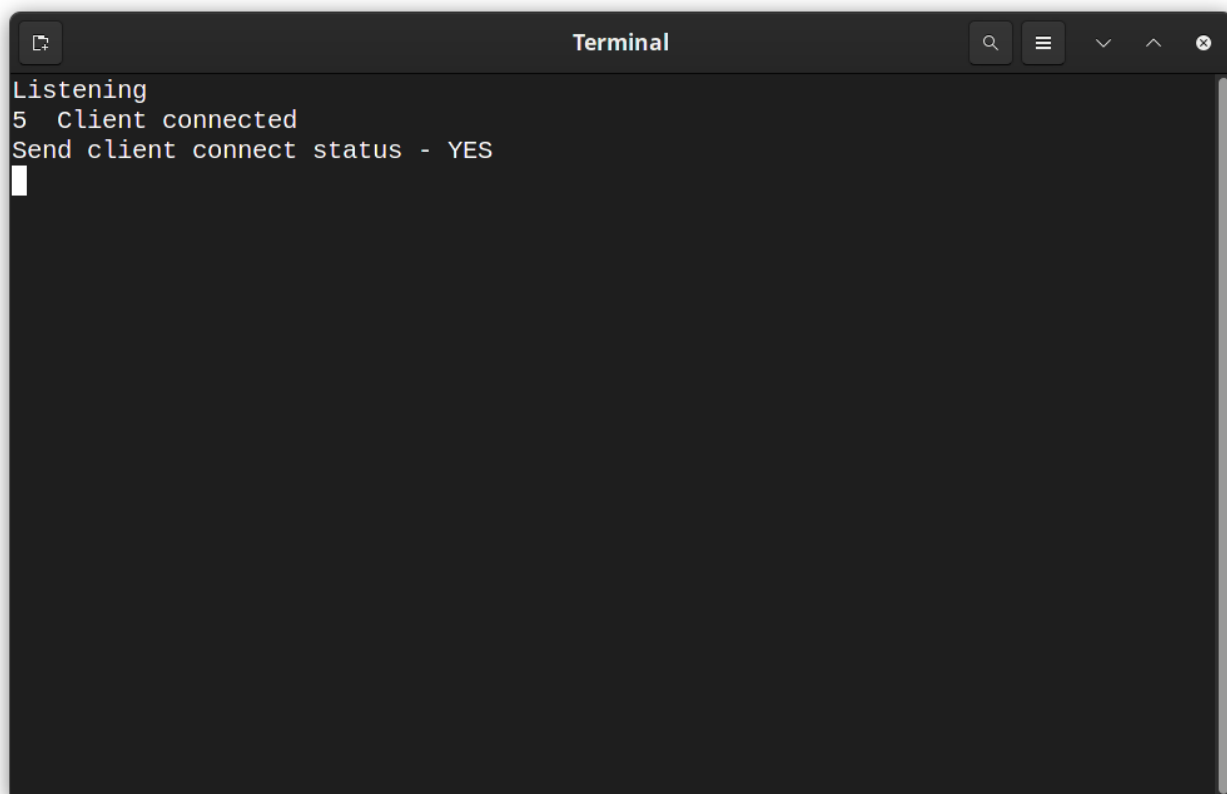
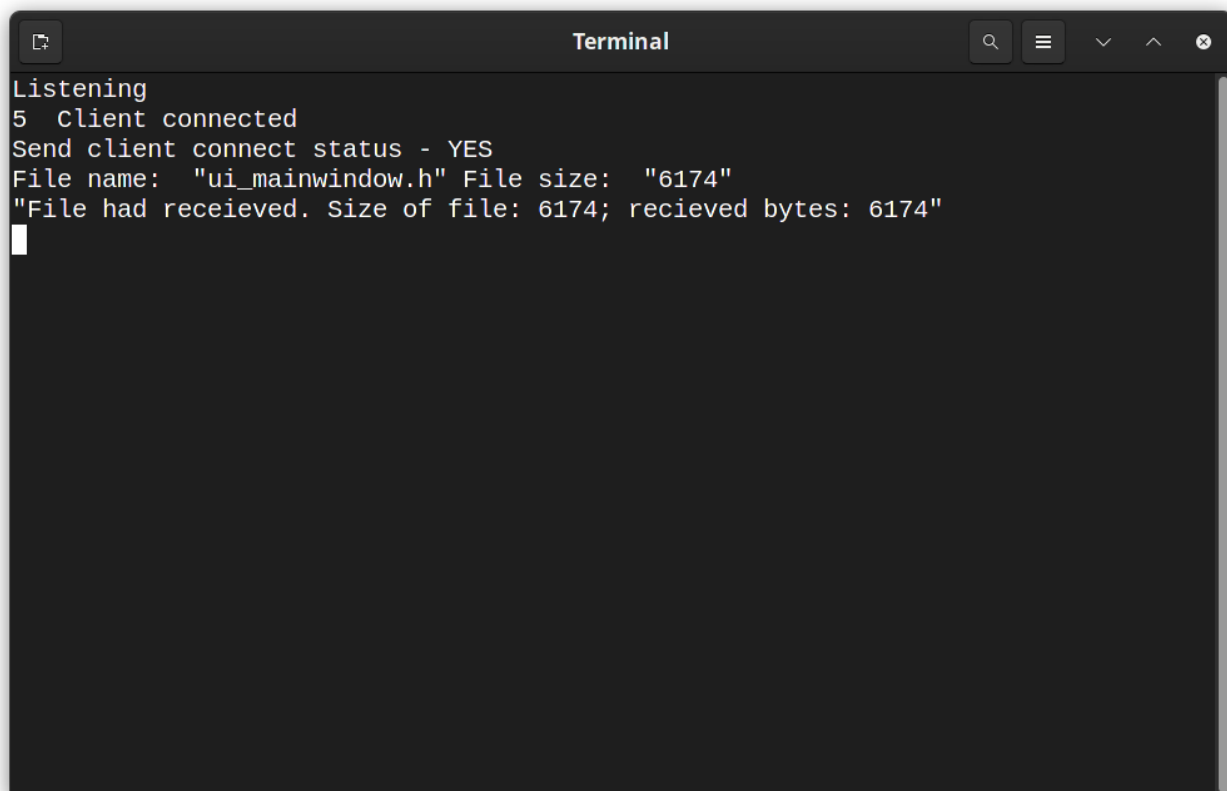


Рис. 3. Запуск сервера

A screenshot of a macOS Terminal window titled "Terminal". The window has a dark background and a light gray title bar with standard macOS window controls (red, yellow, green buttons) and a search icon. The terminal output shows the server is listening, a client has connected, and the server has sent a status message. The cursor is on the line following the status message.

```
Listening
5 Client connected
Send client connect status - YES
█
```

Рис. 4. Подключение клиента к серверу

A screenshot of a macOS Terminal window titled "Terminal". The window has a dark background and a light gray title bar with standard macOS window controls (red, yellow, green buttons) and a search icon. The terminal output shows the server listening, a client connecting, the server sending a status message, and then receiving a file named "ui_mainwindow.h" with a size of 6174 bytes. The cursor is on the line following the file reception message.

```
Listening
5 Client connected
Send client connect status - YES
File name: "ui_mainwindow.h" File size: "6174"
"File had receieved. Size of file: 6174; recieved bytes: 6174"
█
```

Рис. 5. Получение файла от сервера