Министерство Образования, Культуры,

Исследований Республики Молдова

Технический Университет Молдовы

Департамент Программная Инженерия и Автоматика

**Отчёт**

по лабораторной работе №5

**по дисциплине «PR»**

Выполнил: ст.гр. TI-197

Шарафудинов Н.

Проверил: Cernei Irina

Кишинёв - 2022

**Тема:** Приложение клиент-сервер TCP

**Цель работы:** приложения TCP Client Server с помощью Sockets API

**Теория**

1. 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. Надёжность — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений.
2. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.
3. Сокеты бывают блокирующие и неблокирующие. Суть в том, что в случае блокирующих сокетов при попытке прочитать (и записать) данные функция чтения будет ждать до тех пор, пока не прочитает хотя бы один байт или произойдет разрыв соединения или придет сигнал. В случае неблокирующих сокетов функция чтения проверяет, есть ли данные в буфере, и если есть - сразу возвращает, если нет, то она не ждет и также сразу возвращает, что прочитано 0 байт.
4. **accept()**

Ожидает подключения к этому сокету и принимает его.

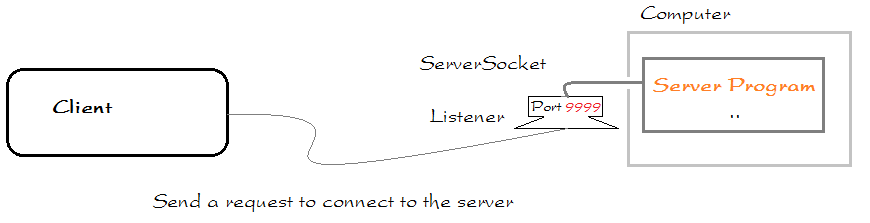
**bind (конечная точка SocketAddress)**

Привязывает ServerSocket к определенному адресу (IP-адрес и номер порта).

**bind (конечная точка SocketAddress, невыполненная работа int)**

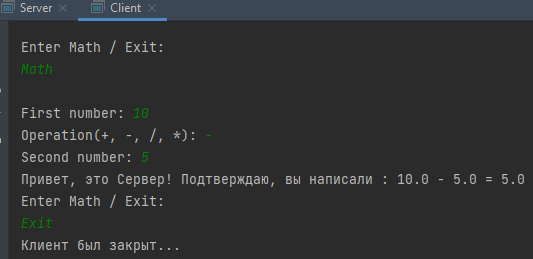
Привязывает ServerSocket к определенному адресу (IP-адрес и номер порта).

1. Нет, recv читает количество байт не больше указанного размера буфера, совершенно не волнуясь по поводу получения всех данных. В лучшем случае с клиента приходит пакет с флагом PSH, намекающий, что имеет смысл отдать данные читателю сейчас.

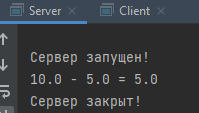


**Результат работы программы**

Client



Server



**Вывод**

В данной лабораторной было рассмотрено как создавать протокол tcp для взаимодействия сервера с клиентом. Так же были внесены изменения в стандартный вид протокола tcp для создания его «эксклюзивности».

**КОД ПРОГРАММЫ:**

Server

import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Locale;  
  
public class Server {  
  
 private static Socket *clientSocket*; //сокет для общения  
 private static ServerSocket *server*; // серверсокет  
 private static BufferedReader *in*; // поток чтения из сокета  
 private static BufferedWriter *out*; // поток записи в сокет  
 private static double *firstNumber*;  
 private static String *operation*;  
 private static double *secondNumber*;  
  
 public static void main(String[] args) {  
 try {  
 try {  
 *server* = new ServerSocket(4004); // серверсокет прослушивает порт 4004  
 System.*out*.println("Сервер запущен!"); // хорошо бы серверу  
 // объявить о своем запуске  
 *clientSocket* = *server*.accept(); // accept() будет ждать пока  
 //кто-нибудь не захочет подключиться  
 try { // установив связь и воссоздав сокет для общения с клиентом можно перейти  
 // к созданию потоков ввода/вывода.  
 // теперь мы можем принимать сообщения  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 // и отправлять  
 *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()));  
  
 while (true) {  
 // ждём пока клиент что-нибудь нам напишет  
 *firstNumber* = Double.*parseDouble*(*in*.readLine());  
 *operation* = *in*.readLine();  
 *secondNumber* = Double.*parseDouble*(*in*.readLine());  
  
 String result = *firstNumber* + " " + *operation* + " " + *secondNumber* + " = " +  
 *solutionResult*(*firstNumber*, *operation*, *secondNumber*);  
 System.*out*.println(result);  
 // не долго думая отвечает клиенту  
 *out*.write("Привет, это Сервер! Подтверждаю, вы написали : " + result + "\n");  
 *out*.flush(); // выталкиваем все из буфера  
 }  
  
 } finally { // в любом случае сокет будет закрыт  
 *clientSocket*.close();  
 // потоки тоже хорошо бы закрыть  
 *in*.close();  
 *out*.close();  
 }  
 } finally {  
 System.*out*.println("Сервер закрыт!");  
 *server*.close();  
 }  
 } catch (Exception e) {  
 System.*err*.println(e.getMessage());  
 }  
 }  
  
 private static double solutionResult(double firstNumber, String operation, double secondNumber) {  
 double result = 0;  
  
 if (operation.equals("/") && secondNumber == 0){  
 System.*out*.println("Stupid user divides by 0");  
 return result;  
 }  
  
 switch (operation){  
 case "+" ->{ result = firstNumber + secondNumber; }  
 case "-" ->{ result = firstNumber - secondNumber; }  
 case "\*" ->{ result = firstNumber \* secondNumber; }  
 case "/" ->{ result = firstNumber / secondNumber; }  
 }  
  
 return result;  
 }  
}

Client

import java.io.\*;  
import java.net.Socket;  
import java.util.Locale;  
  
public class Client {  
  
 private static Socket *clientSocket*; //сокет для общения  
 private static BufferedReader *reader*; // нам нужен ридер читающий с консоли  
 private static BufferedReader *in*; // поток чтения из сокета  
 private static BufferedWriter *out*; // поток записи в сокет  
  
 public static void main(String[] args) {  
 try {  
 try {  
 // адрес - локальный хост, порт - 4004, такой же как у сервера  
 *clientSocket* = new Socket("localhost", 4004); // этой строкой мы запрашиваем  
 // у сервера доступ на соединение  
 *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
 // читать соообщения с сервера  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 // писать туда же  
 *out* = new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream()));  
  
 while (true) {  
  
 System.*out*.println("Enter Math / Exit:");  
 String menu = *reader*.readLine();  
 if (menu.toLowerCase(Locale.*ROOT*).equals("exit"))  
 break;  
  
 // если соединение произошло и потоки успешно созданы - мы можем  
 // работать дальше и предложить клиенту что то ввести  
 // если нет - вылетит исключение  
 System.*out*.print("\nFirst number: ");  
 double firstNumber = Double.*parseDouble*(*reader*.readLine());  
 System.*out*.print("Operation(+, -, /, \*): ");  
 String operation = *reader*.readLine();  
 System.*out*.print("Second number: ");  
 double secondNumber = Double.*parseDouble*(*reader*.readLine());  
  
 // не напишет в консоль  
 *out*.write(firstNumber + "\n"); // отправляем сообщение на сервер  
 *out*.flush();  
 *out*.write(operation + "\n"); // отправляем сообщение на сервер  
 *out*.flush();  
 *out*.write(secondNumber + "\n"); // отправляем сообщение на сервер  
 *out*.flush();  
  
 String serverWord = *in*.readLine(); // ждём, что скажет сервер  
 System.*out*.println(serverWord); // получив - выводим на экран  
 }  
  
 } finally { // в любом случае необходимо закрыть сокет и потоки  
 System.*out*.println("Клиент был закрыт...");  
 *clientSocket*.close();  
 *in*.close();  
 *out*.close();  
 }  
 } catch (Exception e) {  
 System.*out*.print("Trouble with server ");  
 System.*err*.println(e.getMessage());  
 }  
 }  
}