

ЛАБОРАТОРНАЯ РАБОТА №4.
ПО ПРЕДМЕТУ
ПО ПРЕДМЕТУ «ПРОГРАММИРОВАНИЕ СЕТЕВЫХ
ПРИЛОЖЕНИЙ»
ТЕМА: «РАЗРАБОТКА ПРИЛОЖЕНИЙ
АРХИТЕКТУРЫ КЛИЕНТ-СЕРВЕР,
ВЗАИМОДЕЙСТВУЮЩИХ ПО ПРОТОКОЛУ UDP»
1-40 05 01 «Информационные системы и технологии (в бизнес-менеджменте и промышленной безопасности)»

Сетевое взаимодействие по стеку протоколов *TCP/IP* подходит для большинства сетевых задач, которые обеспечивает сериализуемые, предсказуемые и надежные потоки ввода-вывода пакетов пересылаемых данных. Протокол *TCP* включает в себя немало сложных алгоритмов адаптации к перегруженности сетей, а также самые пессимистические предположения относительно потери пакетов, что в итоге может снижать неэффективность способов передачи данных по сети. Альтернативой надежному протоколу передачи данных служат использование **дейтаграмм** – порций данных, передаваемых между участниками сетевого обмена.

Даже если дейтаграмма передается в нужном направлении, то нет гарантий, что она достигнет адресата или адресат окажется на месте, чтобы ее получить. Аналогично, когда дейтаграмма принимается, нет гарантии, что она не была повреждена при передаче по сети или что ее отправитель еще ожидает ответ.

Природа *UDP* как протокола без сохранения состояния полезна при автоматизации серверов, отвечающих на небольшие запросы от огромного числа клиентов, например, *DNS* и потоковых мультимедийных приложений, таких как *IPTV*, *Voice over IP*, протоколы туннелирования *IP* и онлайн-игры.

Дейтаграммы реализуются в платформе *Java* с использованием сетевого протокола *UDP* с помощью двух классов: *DatagramPacket* (пакет данных) и *DatagramSocket* (механизм для передачи и приема пакетов типа *DatagramPacket*).

В классе *DatagramSocket* определяются четыре открытых конструктора.

```
public DatagramSocket() throws SocketException
public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress laddr) throws
SocketException
public DatagramSocket(SocketAddress bindaddr) throws
SocketException
```

Первый конструктор создает объект класса *DatagramSocket*, связанный с любым незанятым портом локального компьютера; второй конструктор создает несвязанный сокет с указанным портом, третий конструктор создает

объект класса *DatagramSocket*, связанный с указанным портом и объектом класса *InetAddress*, четвертый конструктор создает объект класса *DatagramSocket*, связанный с заданным объектом класса *SocketAddress*. Класс *SocketAddress* является абстрактным и реализуется конкретным классом *InetSocketAddress*, который инкапсулирует IP-адрес с номером порта.

Все публичные конструкторы класса *DatagramSocket* могут генерировать исключение типа *SocketException*, если при создании сокета возникают ошибки.

В классе *DatagramSocket* определяется методы *send()* и *receive()*, общие формы которых представлены ниже.

```
public void send(DatagramPacket p) throws IOException
public void receive(DatagramPacket p) throws IOException
```

Метод *send()* отправляет пакет дейтаграммы из сокета *p*. Пакет данных содержит информацию, указывающую данные, подлежащие отправке, их длину, IP-адрес удаленного хоста и номер порта на удаленном хосте.

Метод *receive()* получает пакет дейтаграммы из сокета *p*. Когда этот метод возвращает управление, буфер заполняется полученными данными. Пакет дейтаграммы также содержит отправителя, IP-адрес и номер порта на компьютере отправителя.

В классе *DatagramSocket* определяется также метод *close()*, закрывающий текущий сокет.

```
public void close()
```

Начиная с версии *JDK 7* класс *DatagramSocket* реализует интерфейс *AutoCloseable*, что позволяет управлять сокетом типа *DatagramSocket* в блоке оператора *try with resources*.

```
public class DatagramSocket implements java.io.Closeable {
    //...
}
```

Другие методы из данного класса предоставляют доступ к различным атрибутам, связанным с сокетом *DatagramSocket*. Эти методы перечислены в таблице ниже.

Таблица 1 – Методы из класса *DatagramSocket*

Метод	Описание
<code>public InetAddress getInetAddress()</code>	Возвращает адрес, к которому подключен сокет. Возвращает значение null, если сокет не подключен.
<code>public int getLocalPort()</code>	Возвращает номер порта на локальном хосте, к которому привязан этот сокет, -1, если сокет закрыт, или 0, если он еще не привязан
<code>public int getPort()</code>	Возвращает номер порта, к которому подключен сокет. Возвращает значение -1, если сокет не подключен
<code>public boolean isBound()</code>	Возвращает состояние привязки сокета. Если сокет был привязан до закрытия, то этот метод продолжит

	возвращать значение true после закрытия сокета. Возвращает true, если сокет успешно привязан к адресу.
<code>public boolean isConnected()</code>	Возвращает состояние подключения сокета. Если сокет был подключен до закрытия, то этот метод продолжит возвращать значение true после закрытия сокета. Возвращает true, если сокет успешно подключен к серверу
<code>public void setSoTimeout(int timeout) throws SocketException</code>	Устанавливает SO_TIMEOUT с указанным таймаутом в миллисекундах. Если для этого параметра установлено положительное значение тайм-аута, вызов функции receive() для этого DatagramSocket будет заблокирован только на этот промежуток времени. Если тайм-аут истекает, возникает исключение java.net.SocketTimeoutException, хотя DatagramSocket все еще действителен. Тайм-аут, равный нулю, интерпретируется как бесконечный тайм-аут. Чтобы эта опция вступила в силу, она должна быть включена до ввода операции блокировки. Параметр <i>timeout</i> указывает время ожидания в миллисекундах.

В классе *DatagramPacket* определяется несколько конструкторов. Ниже приведены четыре конструктора данного класса.

```
public DatagramPacket(byte[] buf, int length)
public DatagramPacket(byte[] buf, int offset, int length)
public DatagramPacket(byte[] buf, int offset, int length,
InetAddress address, int port)
public DatagramPacket(byte[] buf, int length, InetAddress
address, int port)
```

Первый конструктор создает *DatagramPacket* для приема пакетов длины *length*. Аргумент длины должен быть меньше или равен *buf.length*. Параметр *buf* определяет буфер для хранения входящей дейтаграммы, параметр *length* – количество байт для чтения.

Второй конструктор создает *DatagramPacket* для приема пакетов длины *length*, указывая смещение в буфере. Аргумент длины должен быть меньше или равен *buf.length*. Параметр *buf* определяет буфер для хранения входящей дейтаграммы, параметр *offset* – смещение в буфере, параметр *length* – количество байт для чтения.

Третий конструктор создает *DatagramPacket* для отправки пакетов длины *length* со смещением *offset* на указанный номер порта на указанном хосте. Аргумент длины должен быть меньше или равен *buf.length*. Параметр *buf* определяет пакетные данные, *offset* – смещение в пакетных данных, *length* – длину пакета данных, *InetAddress* – адрес назначения или null, *port* – номер порта назначения.

Четвертый конструктор создает *DatagramPacket* для отправки пакетов длины *length* на указанный номер порта на указанном хосте. Аргумент длины должен быть меньше или равен *buf.length*. Параметр *buf* определяет пакетные данные, параметр *length* – длину пакета, *InetAddress* – адрес назначения, параметр *port* – номер порта назначения.

Первый и второй конструкторы следует рассматривать как создание и запечатывание письма в конверт без указания адреса, а остальные – как создание и запечатывание письма в конверт и указание адреса на конверте.

В классе *DatagramPacket* определяется несколько методов, которые предоставляют доступ к адресу и номеру порта отдельного пакета, а также к данным и их длине (таблица ниже).

Таблица 2 – Методы из класса *DatagramPacket*

Метод	Описание
<code>public synchronized InetAddress getAddress()</code>	Возвращает IP-адрес компьютера, на который отправляется эта дейтаграмма или с которого она была получена
<code>public synchronized byte[] getData()</code>	Возвращает содержимое буфера, используемого для приема или отправки данных
<code>public synchronized int getLength()</code>	Возвращает длину достоверных данных, которые должны быть отправлены, или длину полученных данных. Эта длина может не полностью совпадать с длиной массива байтов
<code>public synchronized int getOffset()</code>	Возвращает смещение данных, которые должны быть отправлены, или смещение полученных данных.
<code>public synchronized int getPort()</code>	Возвращает номер порта
<code>public synchronized void setAddress(InetAddress iaddr)</code>	Задаёт IP-адрес компьютера, на который отправляется дейтаграмма
<code>public synchronized void setData(byte[] buf)</code>	Устанавливает буфер данных для пакета. Со смещением этого <i>DatagramPacket</i> , установленным на 0, и длиной, установленной на длину <i>buf</i> .
<code>public synchronized void setData(byte[] buf, int offset, int length)</code>	Устанавливает буфер данных для пакета. Задаёт данные (параметр <i>buf</i>), длину (<i>length</i>) и смещение пакета (<i>offset</i>)
<code>public synchronized void setLength(int length)</code>	Устанавливает длину для пакета. Длина пакета – это количество байт из буфера данных пакета, которое будет отправлено, или количество байт из буфера данных пакета, которое будет использоваться для приема данных. Длина должна быть меньше или равна смещению плюс длина буфера пакета
<code>public synchronized void setPort(int iport)</code>	Задаёт номер порта на удаленном хосте, на который отправляется дейтаграмма

В приведенном ниже примере реализуется сетевое взаимодействие клиента с сервером на основе протокола *UDP*. Сообщения передаются по сети со стороны клиента серверу, в ответ клиент получает ответ на отправленную команду.

```
//Example №1. Использование протокола UDP
//Код программы сервера
import java.net.*;
import java.io.*;
class UDPServer {
    public final static int DEFAULT_PORT = 8001;
```

```

        public final String VERSION_COMMAND = "VERSION";
        public final String QUIT_COMMAND = "QUIT";
        public final byte[] VERSION = "V2.0".getBytes();
        public final byte[] UNKNOWN_COMMAND = "Unknown
command".getBytes();
        public void runServer() throws IOException {
            try (DatagramSocket datagramSocket = new
DatagramSocket(DEFAULT_PORT)){
                boolean stopFlag = false;
                System.out.println("UDP Server: Started on " +
InetAddress.getLocalHost().getHostAddress() + ":" +
datagramSocket.getLocalPort());
                while (!stopFlag) {
                    byte[] buf = new byte[512];
                    DatagramPacket recvPacket = new
DatagramPacket(buf, buf.length);
                    datagramSocket.receive(recvPacket);
                    String cmd = new
String(recvPacket.getData()).trim(); //метод trim возвращает
копию строки с пропущенными начальными и конечными пробелами
(т.е. удаляет пробелы в начале и конце строки)
                    System.out.println("UDP Server: Command: " + cmd);
                    DatagramPacket sendPacket = new
DatagramPacket(buf, 0, recvPacket.getAddress(),
recvPacket.getPort());
                    int n = 0; //количество байт в ответе
                    if (cmd.equals(VERSION_COMMAND)) {
                        n = VERSION.length;
                        System.arraycopy(VERSION, 0, buf, 0, n);
                    } else if (cmd.equals(QUIT_COMMAND)) {
                        stopFlag = true; //остановка работы сервера
                    } else {
                        n = UNKNOWN_COMMAND.length;
                        System.arraycopy(UNKNOWN_COMMAND, 0, buf, 0,
n);
                    }
                    sendPacket.setData(buf);
                    sendPacket.setLength(n);
                    datagramSocket.send(sendPacket);
                }
                System.out.println("UDP Server: Stopped");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            UDP Server UDP Server = new UDP Server();
            UDP Server.runServer();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

```

```
//Код программы-клиента
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
class UDPClient {
    public void runClient() throws IOException {
        try (DatagramSocket s = new DatagramSocket()) {
            byte[] buf = new byte[512];
            System.out.println("UDPClient: Started");
            byte[] verCmd = "VERSION".getBytes();
            DatagramPacket sendPacket = new
DatagramPacket(verCmd, verCmd.length,
InetAddress.getByName("127.0.0.1"), 8001);
            s.send(sendPacket);
            DatagramPacket recvPacket = new DatagramPacket(buf,
buf.length);
            s.receive(recvPacket);
            String version = new
String(recvPacket.getData()).trim();
            System.out.println("UDPClient: Server Version: " +
version);

            byte[] quitCmd = "QUIT".getBytes();
            sendPacket.setData(quitCmd);
            sendPacket.setLength(quitCmd.length);
            s.send(sendPacket); //послать данные серверу
            System.out.println("UDPClient: Ended");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        try {
            UDPClient client = new UDPClient();
            client.runClient();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Результаты работы программы (серверная и клиентская часть):

UDPServer	UDPClient
"C:\Program Files\Java\jdk-18.0.1\bin\java.exe"	"C:\Program Files\Java\jdk-18.0.1\bin\java.exe"
UDPServer: Started on 192.168.100.25:8001	UDPClient: Started
UDPServer: Command: VERSION	UDPClient: Server Version: V2.0
UDPServer: Command: QUIT	UDPClient: Ended
UDPServer: Stopped	

Этот пример программы ограничивается обменом данными между портами локальной машины с помощью конструктора класса *DatagramSocket*.

Применение дейтаграмм на компьютере может быть запрещено, например, установленным на нем брандмауэром. Кроме того, номера портов, указанные в программе, могут быть заняты.

Следующий набор классов предназначен для демонстрации надежности использования протокола *UDP* в *Java*. Способ, которым он делает это, заключается в попытке отправить указанное количество сообщений от клиента к серверу. Каждое сообщение содержит сообщение, его порядковый номер и общее количество сообщений, которые должны быть отправлены. Сервер отслеживает полученные сообщения и, когда сообщений больше нет, выводит сводку о количестве полученных сообщений, а также о том, какие из них были потеряны.

```
//Example №2. Использование протокола UDP
//Код приложения сервера
package server;
import java.io.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import common.MessageInfo;
public class UDPServer {
    private DatagramSocket datagramSocket;
    private int port;
    private int totalMessages = -1;
    private int[] receivedMessages;
    private boolean close;
    private void run() throws SocketTimeoutException {
        int packageSize;
        byte[] packageData;
        DatagramPacket datagramPacket;
        //Receive the messages and process them by calling
        processMessage(...)
        System.out.println("Ready for receiving data...");
        while (!close) {
            //Receive request from client
            packageSize = 5000;
            packageData = new byte[5000];
            datagramPacket = new DatagramPacket(packageData,
            packageSize);
            try {
                datagramSocket.setSoTimeout(100000);
                datagramSocket.receive(datagramPacket);
            } catch (IOException e) {
                System.out.println("Error IO exception receiving
                datagramPacket.");
                System.out.println("Could be due to timeout.");
                System.out.println("Now closing server...");
                System.exit(-1);
            }
            processMessage(datagramPacket.getData());
        }
    }
}
```

```

    }
}

public void processMessage(byte[] data) {
    MessageInfo messageInfo = null;
    // Use the data to construct a new MessageInfo object
    ByteArrayInputStream byteStream = new
ByteArrayInputStream(data);
    ObjectInputStream ois;
    try {
        ois = new ObjectInputStream(new
BufferedInputStream(byteStream));
        messageInfo = (MessageInfo) ois.readObject();
        ois.close();
    } catch (ClassNotFoundException e) {
        System.out.println("Error: Could not find class
match for transmitted message.");
    } catch (IOException e) {
        System.out.println("Error: IO exception creating
ObjectInputStream.");
    }
    // On receipt of first message, initialize the receive
buffer
    if (receivedMessages == null) {
        totalMessages = messageInfo.totalMessages;
        receivedMessages = new int[totalMessages];
    }
    // Log receipt of the message
    receivedMessages[messageInfo.messageNumber] = 1;
    // If this ois the last expected message, then identify
any missing messages
    if (messageInfo.messageNumber + 1 ==
messageInfo.totalMessages) {
        close = true;
        String string = "Lost packet numbers: ";
        int count = 0;
        for (int i = 0; i < totalMessages; i++) {
            if (receivedMessages[i] != 1) {
                count++;
                string = string + " " + (i + 1) + ", ";
            }
        }
        if (count == 0) string = string + "None";
        System.out.println("Messages processed...");
        System.out.println("Of " + messageInfo.totalMessages
+ ", " + (messageInfo.totalMessages - count) + " received
successfully...");
        System.out.println("Of " + messageInfo.totalMessages
+ ", " + count + " failed...");
        System.out.println(string);
        System.out.println("Test finished.");
    }
}

public UDPServer(int port) {

```



```

        // Initialize UDP socket for receiving data
        try {
            this.port = port;
            datagramSocket = new DatagramSocket(this.port);
        } catch (SocketException e) {
            System.out.println("Error: Could not create socket
on port " + this.port);
            System.exit(-1);
        }
        // Make it so the server can run
        close = false;
    }
    public static void main(String args[]) {
        int recvPort;
        // Get the parameters from method main
        if (args.length < 1) {
            System.err.println("Error: Arguments required: recv-
port");
            System.exit(-1);
        }
        recvPort = Integer.parseInt(args[0]);
        // Initialize Server object and start it by calling
run()
        UDPServer udpServer = new UDPServer(recvPort);
        try {
            udpServer.run();
        } catch (SocketTimeoutException e) {
        }
    }
}
//Код приложения клиента
package client;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import common.MessageInfo;
public class UDPClient {
    private DatagramSocket datagramSocket;
    public static void main(String[] args) {
        InetAddress serverAddr = null;
        int recvPort;
        int numberOfMessages;
        // Get the parameters of method main
        if (args.length < 3) {
            System.err.println("Arguments required: server
name/IP, recv port, message count");
            System.exit(-1);

```

```

    }
    try {
        serverAddr = InetAddress.getByName(args[0]);
    } catch (UnknownHostException e) {
        System.out.println("Bad server address in UDPClient,
" + args[0] + " caused an unknown host exception " + e);
        System.exit(-1);
    }
    recvPort = Integer.parseInt(args[1]);
    numberOfMessages = Integer.parseInt(args[2]);
    //Construct UDP client class and try to send messages
    System.out.println("Constructing UPD client");
    UDPClient client = new UDPClient();
    System.out.println("Sending messages...");
    client.testLoop(serverAddr, recvPort, numberOfMessages);
}

public UDPClient() {
    //Initialise the UDP socket for sending data
    try {
        datagramSocket = new DatagramSocket();
    } catch (SocketException e) {
        System.out.println("Error creating socket for
sending data.");
    }
}

private void testLoop(InetAddress serverAddr, int recvPort,
int numberOfMessages) {
    MessageInfo messageInfo;
    ByteArrayOutputStream byteStream;
    ObjectOutputStream oos;
    //Sending messages to the server
    for (int i = 0; i < numberOfMessages; i++) {
        messageInfo = new MessageInfo("Data for transferring
",numberOfMessages, i);
        byteStream = new ByteArrayOutputStream(5000);
        try {
            oos = new ObjectOutputStream(new
BufferedOutputStream(byteStream));
            oos.flush();//Flushes the stream. This will
write any buffered output bytes and flush through to the
underlying stream
            oos.writeObject(messageInfo);
            System.out.println(messageInfo);
            oos.flush();
        } catch (IOException e) {
            System.out.println("Error serializing object for
transmission.");
            System.exit(-1);
        }
        //retrieves byte array
        byte[] sendBuf = byteStream.toByteArray();
        send(sendBuf, serverAddr, recvPort);// sending the
data to server
    }
}











```

```

    }
}
private void send(byte[] data, InetAddress destAddr, int
destPort) {
    DatagramPacket datagramPacket;
    //build the datagram packet and send it to the server
    datagramPacket = new DatagramPacket(data, data.length,
destAddr, destPort);
    try {
        datagramSocket.send(datagramPacket);
    } catch (IOException e) {
        System.out.println("Error transmitting packet over
network.");
        System.exit(-1);
    }
}
}
//Код пакета common
package common;
import java.io.Serializable;
public class MessageInfo implements Serializable {
    public static final long serialVersionUID = 52L;
    public int totalMessages;
    public int messageNumber;
    public String data;
    public MessageInfo(String data, int total, int msgNum) {
        this.data=data;
        totalMessages = total;
        messageNumber = msgNum;
    }
    public MessageInfo(String msg) throws Exception {
        String[] fields = msg.split(";");
        if (fields.length!=2)
            throw new Exception("MessageInfo: Invalid string for
message construction: " + msg);
        totalMessages = Integer.parseInt(fields[0]);
        messageNumber = Integer.parseInt(fields[1]);
    }
    public String toString(){
        return new String(data+totalMessages+";"+ messageNumber
+"\n");
    }
}
}

```

Результаты работы программы:

Ready for receiving data...			Constructing UPD client
Messages processed...			Sending messages...
Of 3, 3 received successfully...			Data for transferring 3;0
Of 3, 0 failed...			Data for transferring 3;1
Lost packet numbers: None			Data for transferring 3;2
Test finished.			

Ссылка на проект github: <https://github.com/MarinaJavaIT/Simple-Java-UDP-Example>

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ:

1. Что такое «протокол передачи данных»? Перечислите все известные вам протоколы передачи данных и сферу их применения.
2. Каковы особенности протокола *UDP*?
3. Для чего предназначены классы *DatagramSocket*, *DatagramPacket*?
4. Опишите алгоритм создания клиент-серверного приложения на Java с использованием протокола *UDP*.
5. Каковы различия между протоколами *TCP/IP* и *UDP*?
6. На каком уровне модели *OSI* работают протоколы *TCP/IP* и *UDP*?
7. Напишите сигнатуры методов для приема и отправки дейтаграмм.
8. IP адрес сервиса *google.com* равен 216.58.215.78. Укажите, с помощью какого метода класса *java.net.InetAddress* можно это выяснить?
9. Скорость передачи данных по каналу связи измеряется количеством передаваемых битов информации в секунду?
10. В какой ситуации возникает *SocketTimeoutException*?

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Изучить краткие теоретические сведения лекции и лабораторной работы.
2. Ознакомиться с материалами литературных источников.
3. Ответить на контрольные вопросы лабораторной работы.
4. Разработать алгоритм программы по индивидуальному заданию.
5. Написать, отладить и проверить корректность работы созданной программы.
6. Написать электронный отчет.

Отчет должен быть оформлен по стандарту БГУИР ([Стандарт предприятия СТП 01-2017 "Дипломные проекты \(работы\). Общие требования"](#)) и иметь следующую структуру:

1. титульный лист
2. цель выполнения лабораторной работы
3. теоретические сведения по лабораторной работе
4. формулировка индивидуального задания
5. весь код решения индивидуального задания, разбитый на необходимые типы файлов
6. скриншоты выполнения индивидуального задания
7. выводы по лабораторной работе

ВО ВСЕХ ЗАДАНИЯХ ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН САМ РЕШАТЬ ВЫЙТИ ИЗ ПРОГРАММЫ ИЛИ ПРОДОЛЖИТЬ ВВОД ДАННЫХ. ВСЕ РЕШАЕМЫЕ ЗАДАЧИ ДОЛЖНЫ БЫТЬ РЕАЛИЗОВАНЫ, ИСПОЛЬЗУЯ КЛАССЫ И ОБЪЕКТЫ.

В КАЖДОМ ЗАДАНИИ НЕОБХОДИМО:

Необходимо разработать и протестировать клиент-серверное приложение с использованием протокола *UDP* на языке *Java*. Смоделировать

работу согласно варианту задания. Все параметры заданий необходимо вводить с клавиатуры. Все отправленные и полученные данные записываются в файл на сервере.

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ К ЛАБОРАТОРНОЙ РАБОТЕ:

1. Функционирование клиента и сервера реализовать следующим образом: клиент посылает два числа серверу и одну из математических операций: «*», «/», «+», «-», – сервер соответственно умножает, делит, складывает либо вычитает эти два числа и ответ посылает ответ назад клиенту.

2. Функционирование клиента и сервера реализовать следующим образом: клиент посылает слово серверу, сервер возвращает назад клиенту это слово в обратном порядке следования букв, если встречается спецсимвол, то он заменяется на знак *.

3. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет строку данных серверу, сервер возвращает строку в обратном порядке следования букв, в верхнем регистре, зашифрованную любым способом.

4. Функционирование клиента и сервера реализовать следующим образом: клиент посылает два числа серверу m и n , сервер возвращает $m!+n!$ этих чисел назад клиенту.

5. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет две строки серверу, сервер их сравнивает и возвращает «строки равны», если они одинаковы с учетом регистра, иначе возвращает значение «строки не равны».

6. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет набор латинских букв и спецсимволов (например, !, «, №, \$...) серверу и получает назад символы, упорядоченными по алфавиту, не учитывая спецсимволы.

7. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу произвольный набор символов, сервер замещает каждый четвертый символ на «%», каждый пятый на символ «#».

8. Функционирование клиента и сервера реализовать следующим образом: сервер хранит данные о прогнозе погоды. Клиент отправляет дату и получает соответствующий прогноз.

9. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу произвольные дробные числа и получает назад количество чисел, кратных трем, наибольший и наименьший элементы.

10. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу символьную строку, содержащую пробелы, и получает назад ту же строку, но в ней между словами должен

находиться только один пробел. Сервер также возвращает количество слов в строке.

11. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу две символьные строки. Сервер редактирует строки, удаляя из них идущие друг за другом одинаковые слова. Сервер определяет самое длинное общее слово в полученных заданных строках.

12. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу ФИО. Сервер определяет, находится ли этот человек в списке, хранящемся на сервере, и возвращает все данные об этом человеке.

13. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу два числа и получает «наибольший общий делитель» (наибольший общий делитель) и «наименьшее общее кратное» этих чисел.

14. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу число от 0 до 10 и получает название этого числа прописью на русском, белорусском и английском языках.

15. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу русское слово или фразу и получает перевод с русского языка на английский.

16. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу координаты точки X и Y в декартовой системе координат. Сервер определяет, в какой координатной четверти находится данная точка и отправляет результат клиенту.

17. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу координаты прямоугольной области и точки в декартовой системе координат. Сервер определяет, лежит ли данная точка в прямоугольной области, и отправляет результат клиенту.

18. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу девятизначный номер счета и pin code. Сервер определяет, существует ли такой номер счета в системе и соответствует ли ему указанный pin code. Результат возвращается клиенту.

19. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу ID клиента и кредитный баланс. Сервер определяет, существует ли такой номер счета в системе и анализирует его кредитный баланс. Если кредитный баланс больше указанного клиентом, то возвращается сообщение «Кредит недоступен» иначе «Кредит доступен».

20. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу набор названий в стиле *camelCase*. Сервер возвращает набор слов в стиле *snake_case*.

21. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу текст и ключ «*encryption*» или «*decryption*». Если получен ключ «*encryption*», то сервер реализует шифрование с помощью шифра "Сэндвич": текст разбивается на две одинаковые по количеству символов части, результатом шифрования является текст, в которой символы из первой части чередуются символами из второй части; если получен ключ «*decryption*», то сервер реализует дешифрование. Сервер возвращает полученный зашифрованный текст.

22. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу набор строк, в каждой из которых содержится сначала число, а затем некоторая строка. Выполнить сортировку строк по числу, а строки с одинаковыми числами упорядочить по оставшейся части. Сервер возвращает отсортированный набор.

23. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу набор строк. Сервер возвращает каждую строку в обратном (реверсивном) порядке, изменив расположение символов в каждой строке задом наперёд.

24. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу название алфавита, сервер возвращает клиенту массив символов указанного алфавита (видов алфавитов должны быть обязательно следующие: английский, белорусский, русский).

25. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу шестизначный номер билета. Определить, является ли этот билет «счастливым». «Счастливым» называется такой билет, у которого сумма первых трех цифр равна сумме последних трех.

26. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу количество жителей в государстве и площадь его территории. Сервер рассчитывает и возвращает плотность населения в этом государстве.

27. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу предложение, в котором слова разделены одним пробелом. Сервер определяет и возвращает самое «длинное» слово в предложении.

28. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу строку, содержащую наименование e-mail. Сервер определяет корректен ли формат названия e-mail. Корректность e-mail определяется по следующим правилам: название может состоять из 6–30 знаков и содержать буквы, цифры и символы, может содержать буквы латинского алфавита (a–z), цифры (0–9) и точки (.), запрещено использовать амперсанд (&), знаки равенства (=) и сложения (+), скобки (<>), запятую (,), символ подчеркивания (_), апостроф ('), дефис (-) и несколько точек подряд.

29. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу количество человек. Сервер случайным образом заполняет массив этого размера значениями среднего балла этих человек и возвращает их клиенту.

30. Функционирование клиента и сервера реализовать следующим образом: клиент отправляет серверу число 0 или 1. Если сервер получил число 1, то он заполняет массив таким образом, чтобы значения элементов образовывали убывающую последовательность, если клиент отправил 0, то возрастающую последовательность.