Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**ОТЧЕТ**

Дисциплина: Разработка сетевых приложений

Тема: «Реализация TFTP сервера»

Работу выполнил студент

Группа 3530901/60202

Курякин Д. А.

Преподаватель

Зозуля А. В.

Санкт-Петербург

2020

Оглавление

[**1.** **Цель работы** 3](#_Toc39856916)

[**2.** **Описание протокола** 3](#_Toc39856917)

[**3.** **Описание архитектуры приложения и результаты тестирования** 4](#_Toc39856918)

[**4.** **Вывод** 6](#_Toc39856919)

[**Листинги** 7](#_Toc39856920)

1. **Цель работы**

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции TFTP сервера.

**Основные возможности.** Приложение должно реализовывать следующие функции:

* Прием сообщений от клиента
* Определение запроса на чтение или на запись
* Прием файла в режиме чтения
* Передача файла в режиме запись

**Поддерживаемые команды.** Разработанное приложение должно реализовать следующие команды протокола TFTP:

* GET – запрос на чтение
* PUT – запрос на запись

**Настройка приложения.**

Так как был реализован TFTP север, то из настройки доступно только настройка порта. Настройка размера блока и названия файла осуществляется в ходе запроса клиента.

**Методика тестирования.** Для тестирования приложения использовалась утилита TFTPd64 в качестве TFTP клиента

1. **Описание протокола**
   1. **Основные сведения**

Trivial File Transfer Protocol.

Не поддерживает аутентификацию.

Основан на протоколе UDP, порт 69.

* 1. **Основные команды**
* GET [путь к файлу] – запрос на чтение
* PUT [путь к файлу] – запрос на запись
  1. **Область применения**

Предоставление удаленного доступа к файлам при полном отсутствии средств обеспечения конфиденциальности и безопасности с минимальными усилиями.

1. **Описание архитектуры приложения и результаты тестирования**

В результате работы было сделано приложение на языке Java. Были классы, ServerWRQ для записи файла на сервер, ServerRRQ для чтения файла. Были созданы 6 классов, которые соответствуют каждому пакету протокола: Rrq, Wrq, Data, Ack, Err, Oack.

Программы представляет собой цикл while, в котором при получение запроса на чтение ли запись, программа создает соответствующий поток и в цикле for читает или записывает пока номер блока будет не равен номеру блока который прислал клиент.

Запускаем сервер на порту 69, и отправим клиентом запрос на чтение файла s.AVI с размером блока 16384:

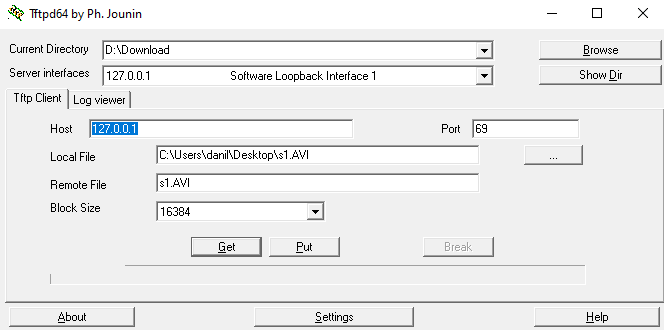


Рисунок 1: Отправка клиентом запрос на чтение.

Server Ready. Port: 69

Read Request from /127.0.0.1

Opcode 1, File name s1.AVI, Mode octet, Block size 16384

bytesOut 1638

…

bytesOut 16384

bytesOut 16352

Transfer completed.(Client /127.0.0.1)

Blocks transfer: 7518

Видно, что сервер отправил 7518 блоков и закончил передачу после передачи блока информации в размере 16352.

Отравим клиентом запрос на запись этогоже файла только сохраним как c1.AVI с размером блока 8192:

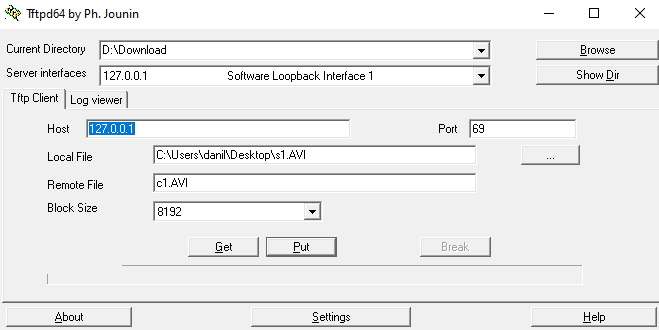


Рисунок 1: Отправка клиентом запрос на запись.

Write Request from /127.0.0.1

Opcode 2, File name s1.AVI, Mode octet, Block size 8192

bytesOut 8192

…

bytesOut 8192

bytesOut 8160

Transfer completed.(Client /127.0.0.1)

Blocks transfer: 15036

Видно, что сервер принял 15036 блоков и закончил прием после блока информации в размере 8160.

1. **Вывод**

В ходе данной работы был разработан TFTP сервер, работает с TFTPd64 клиентом. По результатам тестировании при запросе на чтение сервер отправлял пакеты пока файл не закончился и размер блока не стал меньше того который был в запросе клиента, а клиент при получении пакета меньше размера блока пытался получить слепнувший пакет, что приводило клиент к ошибке. Предположительно это была ошибка клиента. При запросе на чтение все произошло штатно, клиент и сервер закончили передачу и прием при уменьшение размера блока.

# **Листинги**

StartServer.java

|  |
| --- |
| import java.net.\*;  public class StartServer {   public static void main(String argv[]) {  try {   DatagramSocket sock = new DatagramSocket(69);  System.*out*.println("Server Ready. Port: " + sock.getLocalPort());   while (true) {  Packet in = Packet.*receive*(sock, 512);  if (in instanceof Read) {  System.*out*.println("Read Request from " + in.getAddress());  ServerRRQ r = new ServerRRQ((Read) in);  }   else if (in instanceof Write) {  System.*out*.println("Write Request from " + in.getAddress());  ServerWRQ w = new ServerWRQ((Write) in);  }  }  } catch (SocketException e) {  System.*out*.println("Server terminated(SocketException) " + e.getMessage());  } catch (Exception e) {  System.*out*.println("Server terminated(Exception)" + e.getMessage());  }  } } |

ServerRRQ.java

|  |
| --- |
| import java.net.\*; import java.io.\*;   class ServerRRQ extends Thread {   protected DatagramSocket sock;  protected InetAddress host;  protected int port;  protected FileInputStream source;  protected Packet req;  protected String fileName;  protected int blockSize;  protected String mode;  protected String option;  protected long lenFile;   public ServerRRQ(Read request) throws Exception {  try {  req = request;   sock = new DatagramSocket();  sock.setSoTimeout(1000);  fileName = request.fileName();   host = request.getAddress();  port = request.getPort();   String fName = request.get(2, (byte) 0);  mode = request.get(2 + fName.length()+1, (byte) 0);  if (mode.equals("octet")){  option = request.get(2 + fName.length() + 1 + mode.length() + 1, (byte) 0);  blockSize = Integer.*valueOf*(request.get(2 + fName.length() + 1  + mode.length() + 1 + option.length() + 1, (byte) 0));  } else {  blockSize = 512;  }  System.*out*.println("Opcode " + request.get(request.*opOffset*) + ", File name " + fName  + ", Mode " + mode + ", Block size " + blockSize);    File srcFile = new File(fileName);  lenFile = srcFile.length();   if (srcFile.exists() && srcFile.isFile() && srcFile.canRead()) {  source = new FileInputStream(srcFile);  this.start();   } else  throw new Exception("access violation");   } catch (Exception e) {  Error ePak = new Error(1, e.getMessage());  try {  ePak.send(host, port, sock);  } catch (Exception f) {  }   System.*out*.println("Client start failed: " + e.getMessage());  }  }   public void run() {  int timeoutLimit=5;   int bytesRead = blockSize;   int blkNum;   if (req instanceof Read) {  try {  for (blkNum = 1; bytesRead == blockSize; blkNum++) {   Data outPak = new Data(blkNum, source, blockSize);  bytesRead = outPak.getLength()-4;   outPak.send(host, port, sock);   while (timeoutLimit!=0) {  try {   Packet ack = Packet.*receive*(sock, blockSize);  if (!(ack instanceof Ack)){throw new Exception("Client failed");}  Ack a = (Ack) ack;  System.*out*.println("bytesOut " + bytesRead);   if(a.blockNumber()!=blkNum){  throw new SocketTimeoutException("Last packet lost, resend packet");}   timeoutLimit = 5;  break;  }  catch (SocketTimeoutException t) {  System.*out*.println("Send the block again " + blkNum);  timeoutLimit--;  outPak.send(host, port, sock);  }  }  if(timeoutLimit==0){throw new Exception("connection failed");}  }   System.*out*.println("Transfer completed.(Client " +host +")" );  System.*out*.println("Blocks transfer: " + (blkNum-1)); // System.out.println("SHA1 checksum: " + CheckSum.getChecksum(fileName)+"\n");  source.close();  } catch (Exception e) {  Error ePak = new Error(1, e.getMessage());   try {  ePak.send(host, port, sock);  } catch (Exception f) {  }   System.*out*.println("Client failed: " + e.getMessage());  }  }  } } |

ServerWRQ.java

|  |
| --- |
| import java.net.\*; import java.io.\*; import java.nio.charset.Charset;   class ServerWRQ extends Thread {   protected DatagramSocket sock;  protected InetAddress host;  protected int port;  protected FileOutputStream outFile;  protected Packet req;  protected int timeoutLimit = 5;   protected File saveFile;  protected String fileName;  protected int blockSize;  protected String mode;  protected String option;     public ServerWRQ(Write request) throws Exception {  try {  req = request;  sock = new DatagramSocket();  sock.setSoTimeout(1000);   host = request.getAddress();  port = request.getPort();  fileName = request.fileName();   saveFile = new File(fileName);  outFile = new FileOutputStream(saveFile, false);    String fName = request.get(2, (byte) 0);  mode = request.get(2 + fName.length() + 1, (byte) 0);  if (mode.equals("octet")){  option = request.get(2 + fName.length() + 1 + mode.length() + 1, (byte) 0);  blockSize = Integer.*valueOf*(request.get(2 + fName.length() + 1  + mode.length() + 1 + option.length() + 1, (byte) 0));  Oack Oack = new Oack(option, blockSize);  Oack.send(host, port, sock);  } else {  blockSize = 512;  Ack ack = new Ack(0);  ack.send(host, port, sock);  }  System.*out*.println("Opcode " + request.get(request.*opOffset*) + ", File name " + fName  + ", Mode " + mode + ", Block size " + blockSize);   this.start();   } catch (Exception e) {  Error ePak = new Error(1, e.getMessage()); // error code 1  try {  ePak.send(host, port, sock);  } catch (Exception f) {  }   System.*out*.println("Client start failed:" + e.getMessage());  }  }   public void run() {   int blkNum, bytesOut;  if (req instanceof Write) {  try {  for (blkNum = 1, bytesOut = blockSize; bytesOut == blockSize /\*|| bytesOut == 512\*/ ; blkNum++) {  while (timeoutLimit != 0) {  try {  Packet inPak = Packet.*receive*(sock, blockSize);   System.*out*.println();   if (inPak instanceof Error) {  Error p = (Error) inPak;  throw new Exception(p.message());  } else if (inPak instanceof Data) {  Data p = (Data) inPak;   if (p.blockNumber() != blkNum) {  throw new SocketTimeoutException();  }   bytesOut = p.write(outFile);   System.*out*.println("bytesOut " + bytesOut);  Ack a = new Ack(blkNum);  a.send(host, port, sock);  break;  }  } catch (SocketTimeoutException t2) {  System.*out*.println("Time out, resend ack");  Ack a = new Ack(blkNum - 1);  a.send(host, port, sock);  timeoutLimit--;  }  }  if(timeoutLimit==0){throw new Exception("Connection failed");}  }  System.*out*.println("Transfer completed.(Client " +host +")" );  // System.out.println("SHA1 checksum: "+CheckSum.getChecksum(fileName)+"\n");   System.*out*.println("Blocks transfer: " + (blkNum-1));   outFile.close();   } catch (Exception e) {  Error ePak = new Error(1, e.getMessage());  try {  ePak.send(host, port, sock);  } catch (Exception f) {  }   System.*out*.println("Client failed: " + e.getMessage());  saveFile.delete();  }  }  } } |

Ack.java

|  |
| --- |
| final class Ack extends Packet {   protected Ack() {  }   public Ack(int blockNumber) {  length = 4;  this.message = new byte[length];  put(*opOffset*, *ACK*);  put(*blkOffset*, (short) blockNumber);  }   public int blockNumber() {  return this.get(*blkOffset*);  } } |

Data.java

|  |
| --- |
| import java.io.\*;  final class Data extends Packet {   protected Data() {}  public Data(int blockNumber, FileInputStream in, int blockSize) throws IOException {  this.maxData = blockSize;  this.maxPacketLen = this.maxData + 4;  this.length = this.maxPacketLen;  this.message = new byte[this.maxPacketLen];   this.put(*opOffset*, *DATA*);  this.put(*blkOffset*, (short) blockNumber);   this.length = in.read(this.message, this.*dataOffset*, this.maxData) + 4;  }   public int blockNumber() {  return this.get(*blkOffset*);  }   public int write(FileOutputStream out) throws IOException {  out.write(message, *dataOffset*, length - 4);   return (length - 4);  }   public int write(OutputStreamWriter writer) throws IOException {   writer.write(String.*valueOf*(message), *dataOffset*, length - 4);   return (length - 4);  } } |

Error.java

|  |
| --- |
| class Error extends Packet {   protected Error() {  }   public Error(int number, String message) {   length = 4 + message.length() + 1;  this.message = new byte[length];  put(*opOffset*, *ERROR*);  put(*numOffset*, (short) number);  put(*msgOffset*, message, (byte) 0);  }   public int number() {  return this.get(*numOffset*);  }  public String message() {  return this.get(*msgOffset*, (byte) 0);  } } |

Oack.java

|  |
| --- |
| public class Oack extends Packet {   public Oack(){}   public Oack(String option, int sizeBlock) {  String block = String.*valueOf*(sizeBlock);  length = 2 + option.length() + 1 + block.length() + 1;  this.message = new byte[length];  put(*opOffset*, (short) 6);  put(2, option, (byte) 0);  put(2 + option.length() + 1, String.*valueOf*(sizeBlock), (byte) 0);  }   } |

Packet.java

|  |
| --- |
| import java.io.\*; import java.net.\*;  public class Packet {   protected static final short *RRQ* = 1;  protected static final short *WRQ* = 2;  protected static final short *DATA* = 3;  protected static final short *ACK* = 4;  protected static final short *ERROR* = 5;  protected static final short *OACK* = 6;   protected static final int *opOffset*=0;   protected static final int *fileOffset*=2;   protected static final int *blkOffset*=2;  protected static final int *dataOffset*=4;   protected static final int *numOffset*=2;  protected static final int *msgOffset*=4;   public int maxData;  public int maxPacketLen;   protected byte [] message;  protected int length;   protected InetAddress host;  protected int port;   public Packet() {  this.maxData = 512;  this.maxPacketLen = this.maxData + 4;  message=new byte[maxPacketLen];  length= maxPacketLen;  }   public Packet(int maxData) {  this.maxData = maxData;  this.maxPacketLen = this.maxData + 4;  message=new byte[maxPacketLen];  length= maxPacketLen;  }   public static Packet receive(DatagramSocket sock, int blockSize) throws IOException {  Packet in=new Packet(blockSize), retPak=new Packet(blockSize);  DatagramPacket inPak = new DatagramPacket(in.message,in.length);  sock.receive(inPak);   switch (in.get(0)) {  case *RRQ*:  retPak=new Read();  break;  case *WRQ*:  retPak=new Write();  break;  case *DATA*:  retPak=new Data();  break;  case *ACK*:  retPak=new Ack();  break;  case *ERROR*:  retPak=new Error();  break;  case *OACK*:  retPak=new Oack();  break;  }  retPak.message=in.message;  retPak.length=inPak.getLength();  retPak.host=inPak.getAddress();  retPak.port=inPak.getPort();  return retPak;  }   public void send(InetAddress ip, int port, DatagramSocket s) throws IOException {  s.send(new DatagramPacket(message,length,ip,port));  }   public InetAddress getAddress() {  return host;  }   public int getPort() {  return port;  }   public int getLength() {  return length;  }   // Methods to put opcode, blkNum, error code into the byte array 'message'.  protected void put(int startByte, short value) {  message[startByte++] = (byte)(value >>> 8); // first byte  message[startByte] = (byte)(value % 256); // last byte  }   protected void put(int startByte, String value, byte del) {  value.getBytes(0, value.length(), message, startByte);  message[startByte + value.length()] = del;  }   protected int get(int startByte) {  return (message[startByte] & 0xff) << 8 | message[startByte+1] & 0xff;  }   protected String get (int startByte, byte del) {  StringBuffer result = new StringBuffer();  while (message[startByte] != del) result.append((char)message[startByte++]);  return result.toString();  } } |

Read.java

|  |
| --- |
| final class Read extends Packet {   protected Read() {}   public String fileName() {  return this.get(*fileOffset*,(byte)0);  }   public String requestType() {  String fname = fileName();  return this.get(*fileOffset*+fname.length()+1,(byte)0);  } } |

Write.java

|  |
| --- |
| final class Write extends Packet {   protected Write() {}   public String fileName() {  return this.get(*fileOffset*,(byte)0);  }   public String requestType() {  String fname = fileName();  return this.get(*fileOffset*+fname.length()+1,(byte)0);  } } |