Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή

Δίκτυα Υπολογιστών ΙΙ

Θεόδωρος Κατζάλης
AEM:9282
katzalis@auth.gr

May 2, 2021

Contents

1 Δομή του προγράμματος

```
applications
    Copter.java
    Echo.java
    Media.java
    Obd.java
plots
    plot.ipynb
ascii
    audio.txt
    auto.txt
    copter_tcp.txt
    copter.txt
    echo.txt
    https.txt
    image.txt
    obd_tcp.txt
    obd.txt
    temp.txt
    test.txt
    welcome.txt
UserApplication.java
```

- Το αρχείο που βρίσκεται η main είναι το *UserApplication.java*.
- Στον φάκελο *applications*, δημιουργήσαμε ξεχωριστά αρχεία για κάθε εφαρμογή.
- Στον φάκελο *ascii*, βρίσκονται οι έξοδοι του προγράμματος figlet για ascii art λόγους, όπως έχει αναφερθεί και στο report!
- Στον φάκελο *plots*, έχουμε τέλος ένα python αρχείο για να δημιουργήσουμε τα διαγράμματα μας.

2 UserApplication.java

```java
import applications.*;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.System;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.time.LocalDateTime;
import java.util.Scanner;

class UserApplication {

  public static void main(String[] args) {

    printWelcome();

    String[] codes = WebScraping.getCodes();
    int clientPort = Integer.valueOf(codes[0]);
    int serverPort = Integer.valueOf(codes[1]);

    String requestCodeEcho = codes[2];
    String requestCodeImage = codes[3] + "UDP=1024";
    String requestCodeSound = codes[4];
    String requestCodeCopter = codes[5];
    String requestCodeVehicle = codes[6];

    byte[] hostIP = {(byte)155, (byte)207, (byte)18, (byte)208};
    InetAddress hostAddress = null;
    DatagramSocket socket = null;

    try {
      hostAddress = InetAddress.getByAddress(hostIP);
      socket = new DatagramSocket(clientPort);
    } catch (Exception x) {
      x.printStackTrace();
    }

    Scanner in = new Scanner(System.in);

    // control user input
    int flag = 1;
    do {
      System.out.println(
          "\nPlease enter a number (1-11). Available options are:\n"
          + "1) Echo with delay\n"
          + "2) Echo no delay\n"
          + "3) Temperature\n"
          + "4) Image\n"
          + "5) Music\n"
          + "6) Vehicle UDP\n"
          + "7) Ithakicopter UDP\n"
          + "8) Autopilot\n"
          + "9) HTTPS TCP\n"
          + "10) Ithakicopter TCP\n"
          + "11) Vehicle TCP");
      String choiceApp = in.nextLine();

      switch (choiceApp) {
      case "1":

        /* -------------------- Echo with delay ------------------- */
        printASCII("src/ascii/echo.txt");

        try (FileWriter writerInfo =
                 new FileWriter(new File("logs/echo_info_delay.txt"))) {
          writerInfo.write("Info:\n"
                           + "The request code is " + requestCodeEcho + "\n");
          writerInfo.write("Tic: " + LocalDateTime.now() + "\n");

          Echo.telemetry(socket, hostAddress, serverPort, requestCodeEcho,
                         "delay.txt");

          writerInfo.write("Toc: " + LocalDateTime.now());
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      case "2":
        /* -------------------- Echo no delay ------------------- */
        printASCII("src/ascii/echo.txt");

        try (FileWriter writerInfo =
                 new FileWriter(new File("logs/echo_info_no_delay.txt"))) {

          writerInfo.write("Info:\n"
                           + "The request code is "
                           + "requestCodeEcho"
                           + "\n");
          writerInfo.write("Tic: " + LocalDateTime.now() + "\n");

          Echo.telemetry(socket, hostAddress, serverPort, "E0000",
                         "no_delay.txt");
          writerInfo.write("Toc: " + LocalDateTime.now());
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      case "3":
        /* --------------------- Temperature -------------------- */
        printASCII("src/ascii/temp.txt");

        try (FileWriter writerTemp =
                 new FileWriter(new File("logs/temp_info.txt"))) {
          writerTemp.write("Info Temperature app:\n" + requestCodeEcho + "\n" +
                           LocalDateTime.now() + "\n");

          for (int i = 0; i < 1; i++) {
            Echo.execute(socket, hostAddress, serverPort,
                         requestCodeEcho + "T00");
            System.out.println();
          }

          writerTemp.write(LocalDateTime.now() + "\n");
        } catch (Exception x) {
          x.printStackTrace();
        }
        break;

      case "4":
        /* ---------------------------- Image ---------------------------- */
        printASCII("src/ascii/image.txt");

        String encodingImage = "";
        System.out.println("Enter 0 for CAM and 1 for PTZ: ");
        try {
          int userInput = in.nextInt();
          if (Integer.valueOf(userInput) == 1) {
            encodingImage = "CAM=PTZ";
          } else {
            encodingImage = "CAM=FIX";
          }
        } catch (Exception x) {
          x.printStackTrace();
        }

        try (FileWriter writerImage = new FileWriter(
                 new File("logs/image_info_" + encodingImage + ".txt"))) {
          writerImage.write(encodingImage + "\n" + requestCodeImage + "\n" +
                            LocalDateTime.now() + "\n");

          for (int i = 0; i < 1; i++) {
            Media.image(socket, hostAddress, serverPort,
                        requestCodeImage + encodingImage);
            System.out.println();
          }

          writerImage.write(LocalDateTime.now() + "\n");
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      case "5":
        /* ---------------------------- Audio ---------------------------- */
        printASCII("src/ascii/audio.txt");

        String numAudioPackets = "999";

        // song
        String type = "F";

        // tone
        // String type = "T";

        // AQDPCM modulation
        // String encoding = "AQ";

        // DPCM modulation
        String encoding = "";

        // choose song L00 - L??
        String songID = "L02";

        File infoMusic =
            new File("logs/music_info_" + encoding + type + ".txt");
        try (FileWriter writerInfoMusic = new FileWriter(infoMusic)) {
          writerInfoMusic.write(requestCodeSound + "\nEncoding: " + encoding +
                                "\nType: " + type + LocalDateTime.now() + "\n");

          Media.audio(socket, hostAddress, serverPort, encoding, type,
                      numAudioPackets, songID, requestCodeSound);
          System.out.println();

          writerInfoMusic.write(LocalDateTime.now() + "\n");
          writerInfoMusic.close();
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      case "6":
        /* ------------------- Vehicle OBD UDP ------------------ */
        printASCII("src/ascii/obd.txt");
        Obd.udpTelemetry(socket, hostAddress, serverPort, requestCodeVehicle);
        break;

      case "7":
        /* ------------------ Ithakicopter UDP ------------------ */
        printASCII("src/ascii/copter.txt");

        try {
          socket = new DatagramSocket(48078);
        } catch (Exception x) {
          x.printStackTrace();
        }

        copterWelcome();

        try (FileWriter writerCopter =
                 new FileWriter(new File("logs/copter_info.txt"))) {
          writerCopter.write("Info Ithakicopter app:\n" + LocalDateTime.now() +
                             "\n");
          writerCopter.write("MOTOR ALTITUDE TEMPERATURE PRESSURE");

          while (System.in.available() == 0) {
            Copter.udpTelemetry(socket, hostAddress, serverPort, writerCopter);
          }

          writerCopter.write(LocalDateTime.now() + "\n");
        } catch (Exception x) {
          x.printStackTrace();
        }
        break;

      case "8":
        /* ---------------------- Autopilot --------------------- */
        printASCII("src/ascii/auto.txt");

        try (Socket socketAuto = new Socket(hostAddress, 38048)) {
          socket = new DatagramSocket(48078);
          int lowerBound = 160;
```

```java
          int higherBound = 190;
          Copter.autopilot(socket, hostAddress, serverPort, socketAuto,
                           Math.min(200, Math.max(150, lowerBound)),
                           Math.min(200, Math.max(150, higherBound)));
        } catch (Exception x) {
          x.printStackTrace();
          ;
        }

        break;

      case "9":
        /* ---------------------- HTTPS TCP --------------------- */
        printASCII("src/ascii/https.txt");

        try (Socket httpsSocket = new Socket(hostAddress, 80)) {
          https(httpsSocket);
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      case "10":
        /* ------------------- IthakicopterTCP ------------------ */
        printASCII("src/ascii/copter_tcp.txt");

        int target = 180;
        for (int i = 0; i < 10; i++) {
          System.out.println(
              new String(Copter.tcpTelemetry(hostAddress, target)));
        }

        break;

      case "11":
        /* ------------------- Vehicle OBD TCP ------------------ */
        printASCII("src/ascii/obd_tcp.txt");

        try (Socket socketVehicle = new Socket(hostAddress, 29078);
             FileWriter writerVehicleInfo =
                 new FileWriter(new File("logs/car_info.txt"));
             FileWriter writerVehicleData =
                 new FileWriter(new File("logs/car_telemetry.txt"))) {

          writerVehicleInfo.write("Info Vehicle app:\n" + LocalDateTime.now() +
                                  "\n");

          final int minutes = 2;
          final int secondsPerMinute = 60;
          final int timeInterval = minutes * secondsPerMinute;
          float engineTime = 0;
          while (engineTime < timeInterval) {
            engineTime = Obd.tcpTelemetry(socketVehicle, writerVehicleData);
            System.out.println("The engine run time is " + engineTime + "\n");
          }

          writerVehicleInfo.write(LocalDateTime.now() + "\n");
        } catch (Exception x) {
          x.printStackTrace();
        }

        break;

      default:
        System.out.println(
            "Please provide a valid input. If you want to exit then press Control-C.\n");
        flag = 0;
      }
    } while (flag == 0);

    /* ------------------ Close streams ----------------- */
    if (socket != null) {
      try {
        socket.close();
        in.close();
      } catch (Exception x) {
        x.printStackTrace();
      }
    }
    System.out.println("\nShuting down UDP sockets...");

    System.out.println(
        "\nx--------------------Hooray! Java application finished successfully
   !--------------------x");
  }

  /**
   * Print ASCII text
   * @param filePath The path of the file with the ASCII characters to be
   *     printed
   */
  private static void printASCII(String filePath) {
    try {
      Scanner input = new Scanner(new File(filePath));
      while (input.hasNextLine()) {
        System.out.println(input.nextLine());
      }
      Thread.sleep(1500); // pause a little bit to enjoy the view
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  private static void copterWelcome() {
    System.out.println(
        "For Ithakicopter UDP telemetry you need to open ithakicopter.jar");
    System.out.print("Did you open it? If yes press ENTER to continue");
    try {
      System.in.read();
      Thread.sleep(1000); // pause a bit to catch up with the user
      System.out.println("Press ENTER to exit");
      Thread.sleep(1000);
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  /**
   * Print welcome screen ASCII with CYAN color
   */
  private static void printWelcome() {
    // windows users may be not able to view colors on terminal
    final String ANSI_CYAN = "\u001B[36m";
    final String ANSI_RESET = "\u001B[0m";

    try (Scanner input = new Scanner(new File("src/ascii/welcome.txt"))) {
      while (input.hasNextLine()) {
        System.out.print(ANSI_CYAN); // add some color!
        System.out.print(input.nextLine());
        System.out.println(ANSI_RESET);
      }
      System.out.println();
      System.out.print("Press ENTER to continue");
      System.in.read(); // pause a little bit to enjoy the view

    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  private static void https(Socket socket) {
    try {
      InputStream in =
          socket.getInputStream(); // what I receive from the server
      OutputStream out = socket.getOutputStream(); // what i send to the server

      long timeBefore = System.currentTimeMillis();
      out.write(
          "GET /netlab/hello.html HTTP/1.0\r\nHost: ithaki.eng.auth.gr:80\r\n\r\n"
              .getBytes());

      byte[] inputBuffer = in.readAllBytes();
      String message = new String(inputBuffer, StandardCharsets.US_ASCII);
      System.out.println("Ithaki responded via TCP with: \n" + message);
      System.out.println(
          "Time response: " +
          (System.currentTimeMillis() - timeBefore) / (float)1000 + " seconds");
      socket.close();
    } catch (Exception x) {
      x.printStackTrace();
    }
  }
}
```

3 applications

3.1 Echo.java

```java
package applications;

import java.io.File;
import java.io.FileWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.nio.charset.StandardCharsets;

public class Echo {

```

```java
/*
 * UDP TX/RX Echo application with delay
 *
 * WARNING: It doesn't close the DatagramSocket. You should do it manually if
 * it is desired after the call of the function.
 *
 * @param requestCode If request code is set to E000 then the execute will
 *     have no delay for the RX
 */
public static long execute(DatagramSocket socket, InetAddress hostAddress,
                           int serverPort, String requestCode) {
  System.out.println(
      "\n--------------------Echo application--------------------");

  if (requestCode.equals("E0000"))
    System.out.println("Delay: OFF");
  else if (requestCode.length() > 5)
    System.out.println("Mode: Temperature\nDelay: OFF");
  else
    System.out.println("Delay: ON");

  byte[] txbuffer = requestCode.getBytes();
  byte[] rxbuffer = new byte[64];
  long diff = 0;
  try {
    socket.setSoTimeout(3000);
    DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
                                                   hostAddress, serverPort);
    DatagramPacket receivePacket =
        new DatagramPacket(rxbuffer, rxbuffer.length);

    // ACTION
    socket.send(sendPacket);
    System.out.println(
        "The request code is: " + requestCode +
        "\nThe destination port is: " + serverPort +
        "\nMy listening port (clientPort): " + socket.getLocalPort());
    long timeBefore = System.currentTimeMillis();

    // LISTEN
    socket.receive(receivePacket);
    long timeAfter = System.currentTimeMillis();
    diff = timeAfter - timeBefore;
    System.out.println("The time required to reveive a packet is: " + diff +
                       " milliseconds");
    String message =
        new String(receivePacket.getData(), StandardCharsets.US_ASCII);
    System.out.println("Ithaki responded with: " + message);
  } catch (Exception x) {
    x.printStackTrace();
  }
  return diff;
}

/**
 * For a specific time interval send and receive multiple echo packets
 * and 1) calculate throughput with moving average (8 seconds window)
 * ans 2) calculate Retransmission Timeout
 *
 */
```

```java
public static void telemetry(DatagramSocket socket, InetAddress hostAddress,
                             int serverPort, String requestCode,
                             String fileSuffix) {

  FileWriter fileRto = null;
  try (FileWriter fileSamples =
           new FileWriter(new File("logs/echo_samples_" + fileSuffix));
       FileWriter fileThroughput =
           new FileWriter(new File("logs/echo_throughput_" + fileSuffix))) {

    if (!requestCode.equals("E0000"))
      fileRto = new FileWriter(new File("logs/rto.txt"));

    // keep track how many 8 seconds have passed
    int count8sec[] = new int[8];

    float throughput[] = new float[8];

    int cumulativeSum[] = new int[8];

    long tic[] = new long[8];
    long timeBefore = System.currentTimeMillis();
    for (int i = 0; i < 8; i++) {
      tic[i] = timeBefore + i * 1000; // move per second
    }

    int isFirst = 1;
    double rtts = 0;
    double rttd = 0;
    double rto = 0;

    final int minutes = 2;
    final int seconds = 60;
    int timeInterval = seconds * 1000 * minutes;
    while ((System.currentTimeMillis() - timeBefore) < timeInterval) {
      long rtt = Echo.execute(socket, hostAddress, serverPort, requestCode);
      fileSamples.write(rtt + "\n");

      // throughput moving average
      throughputCalc(count8sec, throughput, cumulativeSum, tic, timeBefore,
                     fileThroughput);

      // Retransmission timeout
      if (!requestCode.equals("E0000")) {
        rto(isFirst, rtt, rtts, rttd, rto, fileRto);
        isFirst = 0;
      }
    }
  } catch (Exception x) {
    x.printStackTrace();
  } finally {
    if (fileRto != null) {
      try {
        fileRto.close();
      } catch (Exception x) {
        x.printStackTrace();
      }
    }
  }
}
```

```java

  private static void rto(int isFirst, double rtt, double rtts, double rttd,
                          double rto, FileWriter fileRto) throws Exception {
    // init values
    if (isFirst == 1) {
      rtts = rtt;
      rttd = rtt / 2;
      rto = 1; // according to rfc
      fileRto.write("RTT SRTT RTTd RTO\n");
    }

    // TODO: Fix magic numbers
    double temp = rtts;
    rtts = 0.875 * temp + 0.125 * rtt;

    temp = rttd;
    rttd = 0.75 * temp + 0.25 * Math.abs(rtt - rtts);

    rto = rtts + 1.8 * rttd;
    fileRto.write(rtt + " " + rtts + " " + rttd + " " + rto + "\n");

    System.out.println();
  }

  private static void throughputCalc(int[] count8sec, float[] throughput,
                                     int[] cumulativeSum, long[] tic,
                                     long timeBefore, FileWriter fileThroughput)
      throws Exception {
    for (int i = 0; i < 8; i++) {
      long toc = System.currentTimeMillis() - tic[i];
      System.out.println("The element " + i + " has toc: " + toc);
      if (toc < 8000 && toc > 0) {
        // assume no timeouts during the measurements
        cumulativeSum[i] += 32 * 8;
        System.out.println("Cumsum: " + cumulativeSum[i]);
      } else if (toc > 8000) {
        count8sec[i]++;
        tic[i] = count8sec[i] * 8000 + timeBefore + i * 1000;
        throughput[i] = cumulativeSum[i] / (float)8;
        fileThroughput.write(throughput[i] + "\n");

        System.out.println("I will flush " + cumulativeSum[i] + " cumsum");
        System.out.println("The throughput is: " + throughput[i]);

        cumulativeSum[i] = 0; // let's start again for the next 8 seconds
      }
    }
  }
}
```

3.2 Media.java

```java
package applications;

import java.awt.Desktop;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import javax.sound.sampled.*;

public class Media {

  private static String pathFileImage = "./media/image/ithaki_image.jpg";
  private static String pathFileSound = "./media/music/track.wav";

  public static void image(DatagramSocket socket, InetAddress hostAddress,
                           int serverPort, String requestCode) {
    byte[] txbufferImage = requestCode.getBytes();
    byte[] rxbufferImage = new byte[1024];
    int countPackets = 0;
    long timeBefore = System.currentTimeMillis();

    System.out.println("The request code is " + requestCode);
    DatagramPacket sendPacket = new DatagramPacket(
        txbufferImage, txbufferImage.length, hostAddress, serverPort);
    DatagramPacket receivePacket =
        new DatagramPacket(rxbufferImage, rxbufferImage.length);

    // TX
    try {
      socket.send(sendPacket);
      if (requestCode.contains("DIR")) {
        System.out.println("I am sleeping... Camera needs time to readjust");
        Thread.sleep(5000); // sleep in order for the camera to readjust
      }
    } catch (Exception x) {
      x.printStackTrace();
    }

    // RX
    ByteArrayOutputStream bufferImage = new ByteArrayOutputStream();
  outerloop:
    try {
      socket.setSoTimeout(3000);
      for (;;) {
        socket.receive(receivePacket); // blocking command
        countPackets++;

        for (int i = 0; i < rxbufferImage.length; i++) {
          // System.out.print(String.format("%02X", rxbufferImage[i]));
          bufferImage.write(rxbufferImage[i]); // dynamic byte allocation
          if ((String.format("%02X", rxbufferImage[i]).equals("D9")) &&
              (i != 0)) {
            if ((String.format("%02X", rxbufferImage[i - 1]).equals("FF"))) {
              break outerloop; // stop writing when EOF (0xFFD9 delimiter)
            }
          }
        }
      }
    } catch (Exception x) {
      x.printStackTrace();
    }

    byte[] completeDataImage = bufferImage.toByteArray();
    imageInfo(completeDataImage, countPackets, timeBefore);

    // save image to a file
    saveImage(completeDataImage, pathFileImage);

    // openImage(pathFileImage);
  }

  public static void audio(DatagramSocket socket, InetAddress hostAddress,
                           int serverPort, String encoding, String type,
                           String numAudioPackets, String songID,
                           String requestCodeSound) {

    String completeRequest =
        requestCodeSound + encoding + type + numAudioPackets;
    System.out.println("The request code: " + completeRequest);

    // TX
    byte[] txbufferSound = (songID + completeRequest).getBytes();
    DatagramPacket sendPacket = new DatagramPacket(
        txbufferSound, txbufferSound.length, hostAddress, serverPort);
    try {
      socket.send(sendPacket);
    } catch (Exception x) {
      x.printStackTrace();
    }

    // RX
    byte[] dataSound = new byte[128];
    DatagramPacket receivePacket =
        new DatagramPacket(dataSound, dataSound.length);
    ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
    int countPackets = 0;
    int packetSize = 0;
    long timeBefore = System.currentTimeMillis();

    try (FileWriter writerDiffSamples = new FileWriter(
            new File("logs/" + encoding + type + "diff_samples.txt"));
         FileWriter writerSamples = new FileWriter(
            new File("logs/" + encoding + type + "samples.txt"));
         FileWriter writerMean =
            new FileWriter(new File("logs/aqdpcm_mean.txt"));
         FileWriter writerStep =
            new FileWriter(new File("logs/aqdpcm_step.txt"))) {
      socket.setSoTimeout(3000);
      for (int l = 0; l < Integer.valueOf(numAudioPackets); l++) {
        socket.receive(receivePacket);
        countPackets++;
        packetSize += dataSound.length;

        if (encoding.equals("")) {
          bufferSound.write(dpcm(dataSound, writerDiffSamples, writerSamples));
        } else if (encoding.equals("AQ")) {
          bufferSound.write(adpcm(dataSound, writerDiffSamples, writerSamples,
                                  writerMean, writerStep));
        } else {
          System.out.println("This is not a valid request code");
        }
      }
    } catch (Exception x) {
      x.printStackTrace();
    }
    long timeAfter = System.currentTimeMillis();
    byte[] completeDataSound = bufferSound.toByteArray();
    musicInfo(completeDataSound, timeBefore, timeAfter, countPackets,
              packetSize);

    // only in 16 bit samples does matter. In
    // AQ-DPCM we use 16 bit encoding
    boolean isBigEndian = false;
    int encodingBits = 8;
    if (encoding.equals("AQ")) {
      isBigEndian = true;
      encodingBits = 16;
    }

    AudioFormat modulationPCM =
        new AudioFormat(8000, encodingBits, 1, true, isBigEndian);
    // play sound
    playMusic(completeDataSound, modulationPCM);

    // save music to file
    saveMusic(completeDataSound, modulationPCM);
  }

  private static void openImage(String fileImage) {
    Desktop desktop = Desktop.getDesktop();
    File imageFile = new File(pathFileImage);
    if (imageFile.exists()) {
      try {
        desktop.open(imageFile);
      } catch (Exception x) {
        x.printStackTrace();
      }
    }
  }

  private static void imageInfo(byte[] completeDataImage, int countPackets,
                                long timeBefore) {
    // printImageHex(completeDataImage);
    System.out.println("\nTotal number of packages: " + (countPackets));
    System.out.println("How many Kbytes is the image? " +
                       completeDataImage.length / (float)1000);

    System.out.println("Total amount of time to receive a frame: " +
                       (System.currentTimeMillis() - timeBefore) / (float)1000 +
                       " seconds");
    System.out.println(
        "Total amount of time to receive and write a frame in a .jpg file: " +
        (System.currentTimeMillis() - timeBefore) / (float)1000 + " seconds");
  }

  /**
   * For deubgging purposes print bytes to hexadecimal
   * @param completeData The byte array to be printed as hexadecimal
   */
  private static void printByteHex(byte[] completeData) {
    System.out.println(
        "\nComplete byte content of the data file in hexadecimal format:");
    for (byte i : completeData) {
      System.out.print(String.format("%02X", i));
    }
  }

  private static void musicInfo(byte[] completeDataSound, long timeBefore,
                                long timeAfter, int countPackets,
                                int packetSize) {

    // printByteHex(completeDataSound);

    System.out.println("\n\nTotal number of packages: " + (countPackets));
    System.out.println(
        "How many Kbytes is the sound? " +
        completeDataSound.length / (float)1000 +
        "\nHow many Kbytes is the data that was actually sent? " +
        packetSize / (float)1000);
    System.out.println("Total amount of time to receive sound data: " +
                       (timeAfter - timeBefore) / (float)1000 + " seconds");
  }

  private static void saveImage(byte[] completeDataImage,
                                String pathFileImage) {
    try (FileOutputStream fos = new FileOutputStream(new File(pathFileImage))) {
      fos.write(completeDataImage);
      System.out.println("File has been written successfully");
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  private static void saveMusic(byte[] completeDataSound,
                                AudioFormat modulationPCM) {
    try (AudioInputStream streamSoundInput = new AudioInputStream(
            new ByteArrayInputStream(completeDataSound), modulationPCM,
            completeDataSound.length / modulationPCM.getFrameSize())) {
      AudioSystem.write(streamSoundInput, AudioFileFormat.Type.WAVE,
                        new File(pathFileSound));
      System.out.println("Sound file creation success");
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  private static void playMusic(byte[] completeDataSound,
                                AudioFormat modulationPCM) {
    try (SourceDataLine outputAudio =
            AudioSystem.getSourceDataLine(modulationPCM)) {
      // outputAudio.open(modulationPCM, 3200);
      outputAudio.open(modulationPCM);
      outputAudio.start();

      System.out.println("Getting ready to hear some music?");
      Thread.sleep(2000);
      System.out.print("In 3");
      Thread.sleep(1000);
      System.out.print(", 2");
      Thread.sleep(1000);
      System.out.println(", 1...");
      Thread.sleep(500);
      System.out.println("Listening...");
      Thread.sleep(500);
      outputAudio.write(completeDataSound, 0, completeDataSound.length);
      outputAudio.stop();
      System.out.println("\nSound application success!");
    } catch (Exception x) {
      x.printStackTrace();
    }
  }

  private static byte[] dpcm(byte[] dataSound, FileWriter writerDiffSamples,
                             FileWriter writerSamples) {

    ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
    int init = 0;
    int step = 1;

    for (int i = 0; i < dataSound.length; i++) {
      // get nibbles
      int maskLow = 0x0F;
      int maskHigh = 0xF0;

      // D[i] = x[i] - x[i-1]
      int nibbleLow = dataSound[i] & maskLow;

      // D[i-1] = x[i-1] - x[i-2]
      int nibbleHigh = (dataSound[i] & maskHigh) >> 4;

      // differences
      int diffHigh = (nibbleHigh - 8) * step;
      int diffLow = (nibbleLow - 8) * step;

      // get samples
      int sampleFirst = init + diffHigh;
      int sampleSecond = sampleFirst + diffLow;
      init = sampleSecond;

      // clipping
      int[] samples = {sampleFirst, sampleSecond};
      clipping(samples);

      // write to buffer
      byte[] decodedSound = new byte[2];
      decodedSound[0] = (byte)sampleFirst;
      decodedSound[1] = (byte)sampleSecond;

      try {
        bufferSound.write(decodedSound);
        writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
        writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
      } catch (Exception x) {
        x.printStackTrace();
      }
    }

    try {
      bufferSound.close();
    } catch (Exception x) {
      x.printStackTrace();
    }

    return bufferSound.toByteArray();
```

```java
}

private static byte[] adpcm(byte[] dataSound, FileWriter writerDiffSamples,
                            FileWriter writerSamples, FileWriter writerMean,
                            FileWriter writerStep) {

  // get the header first
  int meanSigned = (dataSound[1] << 8 | dataSound[0]);
  int step = (Byte.toUnsignedInt(dataSound[3]) << 8 |
              Byte.toUnsignedInt(dataSound[2]));
  System.out.println(meanSigned);

  try {
    writerMean.write(meanSigned + "\n");
    writerStep.write(step + "\n");
  } catch (Exception x) {
    x.printStackTrace();
  }

  ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
  // in DPCM we don't know the init value, we assume
  // zero. But here we have data in the header.
  int init = meanSigned;
  for (int i = 3; i < dataSound.length; i++) {
    // get nibbles
    int maskLow = 0x0F;
    int maskHigh = 0xF0;

    // D[i] = x[i] - x[i-1], should be unsigned
    int nibbleLow = (dataSound[i] & maskLow);

    // D[i-1] = x[i-1] - x[i-2], should be unsigned
    int nibbleHigh = (dataSound[i] & maskHigh) >> 4;

    // differences
    int diffHigh = (nibbleHigh - 8) * step;
    int diffLow = (nibbleLow - 8) * step;

    // get samples (implement recursive formula)
    int sampleFirst = init + diffHigh;
    int sampleSecond = sampleFirst + diffLow;
    init = sampleSecond;

    // cliping
    int[] samples = {sampleFirst, sampleSecond};
    clipping(samples);

    // write data to files
    try {
      writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
      writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
    } catch (Exception x) {
      x.printStackTrace();
    }

    // write to buffer
    byte[] decodedSound = new byte[4];
    decodedSound[0] = (byte)(samples[0] >> 8); // MSB of sample 15-8
    decodedSound[1] = (byte)samples[0];         // LSB of sample 7-0
    decodedSound[2] = (byte)(samples[1] >> 8);
```

```java
      decodedSound[3] = (byte)samples[1];

      try {
        bufferSound.write(decodedSound);
      } catch (Exception x) {
        x.printStackTrace();
      }
    }

    return bufferSound.toByteArray();
  }

  private static void clipping(int[] samples) {
    int max16 = (int)(Math.pow(2, 15)) - 1;
    int min16 = -(int)(Math.pow(2, 15));
    for (int j = 0; j < samples.length; j++) {
      if (samples[j] > max16)
        samples[j] = max16;
      else if (samples[j] < min16)
        samples[j] = min16;
    }
  }
}
```

3.3 Obd.java

```java
package applications;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class Obd {

  private static String[] header = {"01 1F", "01 0F", "01 11",
                                    "01 0C", "01 0D", "01 05"};

  public static void udpTelemetry(DatagramSocket socket,
                                  InetAddress hostAddress, int serverPort,
                                  String requestCode) {

    byte[] rxbuffer = new byte[16];
    DatagramPacket receivePacket =
        new DatagramPacket(rxbuffer, rxbuffer.length);

    for (int i = 0; i < header.length; i++) {
      // TX
      String completeCode = (requestCode + "OBD=" + header[i]);
      byte[] txbuffer = completeCode.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
                                                     hostAddress, serverPort);
      System.out.println("Complete request: " + completeCode);
      try {
        socket.send(sendPacket);
      } catch (Exception x) {
        x.printStackTrace();
      }
      long timeBefore = System.currentTimeMillis();

      // RX
      try {
        socket.setSoTimeout(3000);
        socket.receive(receivePacket);
        String message = new String(rxbuffer, StandardCharsets.US_ASCII);
        System.out.println("Ithaki responded via UDP with: " + message);
        System.out.println("Ithaki UDP time response: " +
                           (System.currentTimeMillis() - timeBefore) /
                               (float)1000 +
                           " seconds");

        int[] values = parser(message);
        formula(values[0], values[1], header[i]);
      } catch (Exception x) {
        x.printStackTrace();
      }
    }
  }

  public static float tcpTelemetry(Socket socket, FileWriter writerVehicle) {
    float engineTime = 0;

    try {
      InputStream in = socket.getInputStream();
      OutputStream out = socket.getOutputStream();
      BufferedReader bf = new BufferedReader(new InputStreamReader(in));

      for (int i = 0; i < header.length; i++) {
        out.write((header[i] + "\r").getBytes());
        // out.flush();
        long timeBefore = System.currentTimeMillis();
        System.out.println(
            "Created TCP socket and set output stream... Waiting for response");

        System.out.println("Header: " + header[i]);
        String data = bf.readLine();
        System.out.println("Ithaki responded via TCP with: " + data);
        System.out.println("Ithaki TCP time response: " +
                           (System.currentTimeMillis() - timeBefore) /
                               (float)1000 +
                           " seconds");

        int[] values = parser(data);
        float value = formula(values[0], values[1], header[i]);
        writerVehicle.write(value + " ");
        if (header[i] == "01 1F")
          engineTime = value;
      }

      writerVehicle.write("\n");
    } catch (Exception x) {
      x.printStackTrace();
    }

    return engineTime;
  }
```

```java

private static float formula(int first, int second, String header) {
  float value = 0;
  switch (header) {
  case "01 1F":
    int engineRunTime = first * 256 + second;
    System.out.println("Engine run time: " + engineRunTime);
    value = engineRunTime;
    break;

  case "01 0F":
    int intakeAirTemp = first - 40;
    System.out.println("Intake Air Temperature: " + intakeAirTemp);
    value = intakeAirTemp;
    break;

  case "01 11":
    float throttlePos = (first * 100) / (float)255;
    System.out.println("Throttle position: " + throttlePos);
    value = throttlePos;
    break;

  case "01 0C":
    float engineRpm = ((first * 256) + second) / (float)4;
    System.out.println("Engine RPM: " + engineRpm);
    value = engineRpm;
    break;

  case "01 0D":
    int speed = first;
    System.out.println("Vehicle speed: " + speed);
    value = speed;
    break;

  case "01 05":
    int coolantTemp = first - 40;
    System.out.println("Coolant Temperature: " + coolantTemp);
    value = coolantTemp;
    break;

  default:
    System.out.println(
        "Something went wrong calculating formual for vehicle stats");
  }
  System.out.println();
  return value;
}

private static int[] parser(String data) {
  String byte1 = data.substring(6, 8);
  int first = Integer.parseInt(byte1, 16);
  String byte2 = "";
  int second = 0;
  if (data.length() > 8) {
    byte2 = data.substring(9, 11);
    second = Integer.parseInt(byte2, 16);
  }
  System.out.println();

  int[] temp = {first, second};
```

```java
    return temp;
  }
}
```

3.4 Copter.java

```java
package applications;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.nio.charset.StandardCharsets;

public class Copter {
  public static String udpTelemetry(DatagramSocket socket,
                                    InetAddress hostAddress, int serverPort,
                                    FileWriter writerCopter) {
    // TX
    // open ithakicopter.jar

    // RX only
    byte[] rxbuffer = new byte[128];
    DatagramPacket receivePacket =
        new DatagramPacket(rxbuffer, rxbuffer.length);

    String telemetry = "";
    try {
      socket.setSoTimeout(3000);
      socket.receive(receivePacket);
      telemetry = new String(rxbuffer, StandardCharsets.US_ASCII);
      System.out.println("Received data via UDP: " + telemetry);

      String[] tokensMotor = telemetry.split("LMOTOR=");
      String[] tokensAltitude = telemetry.split("ALTITUDE=");
      String[] tokensTemp = telemetry.split("TEMPERATURE=");
      String[] tokensPress = telemetry.split("PRESSURE=");
      writerCopter.write(tokensMotor[1].substring(0, 3) + " ");
      writerCopter.write(tokensAltitude[1].substring(0, 3) + " ");
      writerCopter.write(tokensTemp[1].substring(1, 6) + " ");
      writerCopter.write(tokensPress[1].substring(0, 7) + "\n");
    } catch (Exception x) {
      x.printStackTrace();
    }
    return telemetry;
  }

  public static String tcpTelemetry(InetAddress hostAddress, int target) {
    String telemetry = "";
    try (Socket socket = new Socket(hostAddress, 38048)) {
      InputStream in = socket.getInputStream();
      OutputStream out = socket.getOutputStream();
      BufferedReader bf = new BufferedReader(new InputStreamReader(in));
      ByteArrayOutputStream bos = new ByteArrayOutputStream();

      String command = "AUTO FLIGHTLEVEL=" + target + " LMOTOR=" + target +
                       " RMOTOR=" + target + " PILOT \r\n";
      // System.out.print("Request: " + command);
      out.write(command.getBytes());
      out.flush();

      // skip telemetry info
      for (int i = 0; i < 14; i++) {
        bos.write((bf.readLine() + "\n").getBytes());
      }
      String data = new String(bos.toByteArray(), StandardCharsets.US_ASCII);
      // System.out.println("Received data via TCP: " + data);

      String[] tokens = data.split("\n");
      // take only the useful data and skip the info ithaki sent
      telemetry = tokens[13];
    } catch (Exception x) {
      x.printStackTrace();
    }
    return telemetry;
  }

  /*
   * tcpTelemetry function for the TX and udpTelemetry for RX. The way that
   * these two functions are implemented force the autopilot to be used with a
   * combination of these two. We want to send a command only if it is needed
   * and we want to listen all the time to get feedback.
   *
   * Notes: Work In Progress
   */
  public static void autopilot(DatagramSocket listen, InetAddress hostAddress,
                               int serverPort, Socket send, int lowerBound,
                               int higherBound) {

    lowerBound = Math.min(lowerBound, higherBound);
    higherBound = Math.max(lowerBound, higherBound);

    int target = (lowerBound + higherBound) / 2;
    int motor = -1;

    try {
      System.out.println("AUTOPILOT: ON");
      System.out.println(
          "You need to open ithakicopter.jar. Press ENTER to continue...");
      System.in.read();
      System.out.println("Press Control-C to exit...");
      Thread.sleep(1000);
      for (;;) {
        if ((motor < (lowerBound)) || (motor > (higherBound))) {
          System.out.println("Send packet. Readjust...");
          tcpTelemetry(hostAddress, target);
        }

        String telemetry =
            Copter.udpTelemetry(listen, hostAddress, serverPort, null);
        String[] tokens = telemetry.split("LMOTOR=");
        motor = Integer.parseInt(tokens[1].substring(0, 3)); // get motor values

        System.out.println("Parsed motor values: " + motor);
      }
    } catch (Exception x) {
      x.printStackTrace();
      System.out.println("AUTOPILOT failed");
    }
  }
}
```