**POLITECHNIKA POZNAŃSKA**

WYDZIAŁ ELEKTRYCZNY

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

FILIP WICHA, MARCIN WIKTOROWSKI

Sprawozdanie z zajęć laboratoryjnych

**PROGRAMOWANIE SIECIOWE – PROTOKOŁY BINARNE**

9 grudnia 2017

**1. TREŚĆ ZADANIA**

1. Temat: Komunikacja pomiędzy klientem a serwerem (1:1), w oparciu o autorski protokół binarny.

Protokół:

* połączeniowy,
* wszystkie dane przesyłane w postaci binarnej,
* pole operacji o długości 3 bitów,
* pola liczb o długości 32 bitów,
* pole statusu o długości 2 bitów,
* pole identyfikatora o długości 8 bitów,
* dodatkowe pola zdefiniowane przez programistę.

Funkcje oprogramowania:

* nawiązanie połączenia,
* uzgodnienie identyfikatora sesji,
* wykonywanie operacji matematycznych na dwóch argumentach:
  + 000 – dodawanie,
  + 001 – odejmowanie,
  + 010 – mnożenie,
  + 011 – dzielenie,
  + 4 inne, zdefiniowane przez programistę.
* wykonywanie operacji matematycznych na wielu argumentach:
  + (przykład) o klient przesyła wartość liczbową, operator oraz informację, czy jest to ostatni komunikat,
  + serwer jawnie potwierdza otrzymanie każdego komunikatu i zwraca wyniki pośrednie (o ile możliwe).
  + zakończenie połączenia.

Inne:

* gdy wartość wyniku wykracza poza zakres zmiennej, powinien zostać zwrócony kod błędu,
* identyfikator sesji powinien być przesyłany w trakcie komunikacji.

**2. OPIS PROTOKOŁU**

A. Format komunikatu

Komunikat zawiera 6 pól:

1. operation:
   1. pole zawierające informacje o operacji w postaci 3 bitów:
      1. dla operacji dwuargumentowej:
         1. 000 – dodawanie,
         2. 001 – odejmowanie,
         3. 010 – mnożenie,
         4. 011 – dzielenie,
         5. 100 – obliczanie miejsca zerowego funkcji zerowej,
         6. 101 – obliczanie logarytmu,
         7. 110 – obliczanie średniej liczb,
         8. 111 – określanie czy liczby są takie same.
      2. dla operacji wieloargumentowej:
         1. 000 – dodawanie,
         2. 001 – odejmowanie,
         3. 010 – mnożenie,
         4. 011 – dzielenie,
         5. 100 – kończenie działań (symbol „=”).
2. number1:
   1. pole zawierające liczbę 32 bitową;
3. number2:
   1. pole zawierające liczbę 32 bitową;
4. state:
   1. pole zawierające informację o możliwych błędach w postaci 2 bitów:
      1. 00 – brak błędu,
      2. 01 – dzielenie przez 0,
      3. 10 – przekroczenie zakresu int32.
5. sessionId:
   1. pole zawierające informacje o numer sesji w postaci 8 bitów;
6. mode:
   1. pole zawierające informacje o sposobie obliczania w postaci 2 bitów:
      1. 00 – operacja dwuargumentowa
      2. 01 – operacja wieloargumentowa, ale nie ostatni pakiet,
      3. 10 – operacja wieloargumentowa, ostatni pakiet,

B. Zbiór komend i odpowiedzi

Komendy:

* void Serialize()– funkcja, która serializuje dane z obiektu Packet do tablicy bitów;
* void Deserialize() – funkcja, która deserializuje dane z tablicy bajtów do obiektu Packet;
* byte[] GetBytes() – funkcja, która wykonuje funkcję Serialize() oraz zamienia tablicę bitów na tablicę bajtów;

**3. APLIKACJA UŻYTKOWNIKA ORAZ APLIKACJA SERWERA**

W naszej implementacji aplikacja użytkownika (klienta) oraz serwera jest połączona. W pierwszym menu wybiera się czy aplikacja ma się zachowywać jak klient czy jak serwer.

Klasa Program:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ProjectTS  {  class Program  {  static Server server; //create server  static Client client; //create client    static void Main(string[] args)  {  Console.WriteLine("Start as:\n1.Client\n2.Server"); //menu that allows to choose if the program is client or server  if(Convert.ToInt32(Console.ReadLine()) == 1)  {  client = new Client();  if (client.Connect())  {  client.Run();  client.Disconnect();  }  }  else  {  server = new Server();  server.Startup();  }  Console.ReadLine();  }  }  } |

Klasa Client:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Threading.Tasks;  namespace ProjectTS  {  public class Client  {  const int DEFAULT\_PORT = 211;  Mode mode = Mode.NotDefined; //flag that informs which mode is in use  int currentSessionId; //flag that informs about current session id  bool isConnected = false; //flag that informs about connection  Socket clientSocket; //create client's socket    List<string> operands = new List<string>(new string[] { "+", "-", "\*", "/", "x +", "log", "average", "==", "=",});  string \_equation = "";  string equation  {  get  {  return \_equation;  }  set  {  \_equation = value + " ";  displayMenu();  }  }  public bool Connect()  {  IPAddress serverAddr = IPAddress.Parse("127.0.0.1"); //set IP of a server  var clientEndPoint = new IPEndPoint(serverAddr, DEFAULT\_PORT); //set an endpoint  clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);  //Try to connect to server (Timeout = 30s)  int Timeout = 6;  while(Timeout>0){  try  {  clientSocket.Connect(clientEndPoint); //connect to server  isConnected = true;  Timeout = 0;  }  catch(Exception ex)  {  Console.WriteLine(ex.Message);  Timeout--;  System.Threading.Thread.Sleep(5000);  if (Timeout == 0)  {  Console.WriteLine("Connection canceled due to timeout");  return false;  }  }  }  Console.WriteLine("Connected to " + clientSocket.LocalEndPoint.ToString());  GetSessionId(); //get sessionId from server  return true;  }  public bool Disconnect()  {  clientSocket.Disconnect(false);  return true;  }  public void Run()  {  while (isConnected)  {  SendData();  ReceiveData();  if(mode == Mode.MultiArgumentsLP)  {  return; //when server sets mode of packet as MultiArgumentsLP (that it is last packet),  } //while(isConnected) stops  }  }  private void displayMenu()  {  Console.Clear();  Console.WriteLine("Mode: " + mode.ToString());  Console.WriteLine(\_equation);  }  void GetSessionId()  {  byte[] buffer = new byte[256];  try  {  var bytesrecd = clientSocket.Receive(buffer); //get firstt packet fom server  Packet pack = new Packet(buffer);  currentSessionId = pack.sessionId; //set current sessionId  Console.WriteLine("Session ID is set: " + currentSessionId);  }  catch (Exception ex)  {  Console.WriteLine(ex.ToString());  }  }  private void ReceiveData()  {  byte[] buffer = new byte[256];  var bytesrecd = clientSocket.Receive(buffer); //receive packet from server  Packet pack = new Packet(buffer); //deserialize and create packet with received data  switch (pack.state)  {  case State.Nothing:  if (mode == Mode.MultiArguments)  {  Console.WriteLine("Actual result: " + pack.number1);  }  else  {  equation += pack.number1;  }  break;  case State.DivisionByZero:  equation = "Division by 0";  break;  case State.OverFlow:  equation = "Out of range! ";  break;  }  }  private void SendData()  {  bool Error = true;  do  {  Packet pack = new Packet(); //create packet to send  try  {  if (mode == Mode.NotDefined) //if mode is not defined, you firstly you have to set mode  {  Console.WriteLine("Choose the method:\n1.Two argument\n2.Multiargument\n");  mode = (Mode)Convert.ToInt32(Console.ReadLine()) - 1;  displayMenu();  }  if (mode == Mode.TwoArguments) //if mode is set to TwoArguments get operand x, y and choose operation  {  pack.mode = mode;  Console.Write("Give an operand x: ");  pack.number1 = Convert.ToInt32(Console.ReadLine());  equation = pack.number1.ToString();  Console.WriteLine("Choose operation:\n1.Addition\n2.Substraction\n3.Multiplication\n4.Division\n5.Linear Function\n6.Log\n7.Average\n8.Equals");  pack.operation = (Operation)Convert.ToInt32(Console.ReadLine()) - 1;  equation += operands[(int)pack.operation];  Console.Write("Give an operand y: ");  pack.number2 = Convert.ToInt32(Console.ReadLine());  equation += pack.number2 + " " + operands[8];  }  else if (mode == Mode.MultiArguments) //if mode is set to TwoArguments get operand x and choose operation  {  pack.mode = mode;  Console.Write("Give an operand x: ");  pack.number1 = Convert.ToInt32(Console.ReadLine());  equation += pack.number1;  pack.number2 = 0;  Console.WriteLine("Choose operation:\n1.Addition\n2.Substraction\n3.Multiplication\n4.Division\n5.Equals");  pack.operation = (Operation)Convert.ToInt32(Console.ReadLine()) - 1;  if (pack.operation == Operation.LinearFunction)  {  equation += operands[8];  pack.operation = Operation.Equals;  pack.mode = mode = Mode.MultiArgumentsLP;  }  else  {  equation += operands[(int)pack.operation];  }  }  pack.state = State.Nothing; //set state  pack.sessionId = currentSessionId; //set sessionId  }  catch (Exception ex)  {  Console.WriteLine(ex.ToString());  continue;  }  try  {  clientSocket.Send(pack.GetBytes()); //serialize and send data to server  Error = false;  }  catch (Exception ex)  {  Console.WriteLine(ex.ToString());  isConnected = false;  }  } while (Error);  }  }  } |

Klasa Server:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Net;  using System.Net.Sockets;  namespace ProjectTS  {  public class Server  {  const string DEFAULT\_SERVER = "localhost";  const int DEFAULT\_PORT = 211;  bool isConnected = false;  //Sockets  Socket serverSocket;  Socket clientSocket;  int currentSessionId; //flag that informs about current session Id  public int result = 0; //variable that holds current result of calculation  Operation operation = Operation.Addition; // first operation is set to addition,  State state = State.Nothing; //because the first number form first packet must be set as current result  public void Startup()  {  IPAddress serverAddr = IPAddress.Parse("127.0.0.1"); //set client's IP  serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp); //create server socket  serverSocket.Bind(new IPEndPoint(serverAddr, DEFAULT\_PORT)); //bind  Console.WriteLine("Server started at:" + serverSocket.LocalEndPoint.ToString());  int backlog = 0; //how many pending connections the queue will hold  try  {  serverSocket.Listen(backlog); //start listening  Console.WriteLine("Server listening");  clientSocket = serverSocket.Accept(); //accept connection  isConnected = true;  Console.WriteLine("Client connected");  }  catch (Exception ex)  {  Console.WriteLine("Failed to listen" + ex.ToString());  }  SetSessionId(); //set sessionId  this.Run();  }  public void Run()  {  while (isConnected)  {  ReceiveData();  SendData();  }  }  void SetSessionId()  {  Packet pack = new Packet(); //create packet with Id to be set by client  pack.sessionId = currentSessionId = 7; //set Id to 7  clientSocket.Send(pack.GetBytes()); //serialize and send  Console.WriteLine("Session ID is set: " + currentSessionId);  }  public void ReceiveData()  {  byte[] buffer = new byte[256];  try  {  var bytesrecd = clientSocket.Receive(buffer); //receive data from client  Packet pack = new Packet(buffer); //deserialize and create packet with received data  if (pack.mode == Mode.TwoArguments) //if mode is two arguments, calculate with chosen operation  {  state = State.Nothing;  switch (pack.operation)  {  case Operation.Addition:  result = checked(pack.number1 + pack.number2);  break;  case Operation.Substraction:  result = checked(pack.number1 - pack.number2);  break;  case Operation.Multiplication:  result = checked(pack.number1 \* pack.number2);  break;  case Operation.Division:  if (pack.number2 == 0)  {  state = State.DivisionByZero;  break;  }  result = checked(pack.number1 / pack.number2);  break;  case Operation.LinearFunction:  if (pack.number1 == 0)  {  state = State.DivisionByZero;  break;  }  result = checked((-pack.number2) / pack.number1);  break;  case Operation.Log:  result = checked(Convert.ToInt32(Math.Log(pack.number1, pack.number2)));  break;  case Operation.Average:  result = checked(((pack.number1 + pack.number2) / 2));  break;  case Operation.Equals:  if (pack.number1 == pack.number2) result = 1;  else result = 0;  break;  }  }  else if (pack.mode == Mode.MultiArguments || pack.mode == Mode.MultiArgumentsLP) //if mode is multi arguments, calculate with chosen operation  {  switch (operation)  {  case Operation.Addition:  result = checked(result + pack.number1);  break;  case Operation.Substraction:  result = checked(result - pack.number1);  break;  case Operation.Multiplication:  result = checked(result \* pack.number1);  break;  case Operation.Division:  if (pack.number1 == 0)  {  state = State.DivisionByZero;  }  else  {  result = checked( result / pack.number1);  }  break;  }  if(pack.operation == Operation.Equals)  {  isConnected = false;  }  operation = pack.operation;  }  }  catch (System.OverflowException e)  {  state = State.OverFlow;  result = 0;  }  catch (Exception ex)  {  Console.WriteLine(ex.ToString());  }  Console.WriteLine("Server received packet");  }  public void SendData()  {  Packet pack = new Packet(); //create empty packet  pack.operation = Operation.Addition; //set any operation  pack.number1 = result; //first number in packet will be the result of calculation to this point  pack.number2 = 0; //set any second number  pack.state = state; //set state to that from flag  pack.sessionId = currentSessionId; //set Id from flag  pack.mode = Mode.NotDefined; //set any mode    try  {  clientSocket.Send(pack.GetBytes()); //serialize and send  }  catch (Exception ex)  {  Console.WriteLine(ex.ToString());  isConnected = false;  }  }  }  } |

Klasa Packet:

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ProjectTS  {  #region Enums  public enum Operation  {  Addition = 0b000,  Substraction = 0b001,  Multiplication = 0b010,  Division = 0b011,  LinearFunction = 0b100,  Log = 0b101,  Average = 0b110,  Equals = 0b111  }  public enum Mode  {  TwoArguments = 0b00,  MultiArguments = 0b01,  MultiArgumentsLP = 0b10,  NotDefined = 0b011  }  public enum State  {  Nothing = 0b00,  DivisionByZero = 0b01,  OverFlow = 0b10,  NotDefined = 0b11  }  #endregion  public class Packet  {  BitArray bitArr; //temporary array of bits    public Operation operation;  public int number1 = 0;  public int number2 = 0;  public State state = State.Nothing;  public int sessionId;  public Mode mode;  public Packet()  {  bitArr = new BitArray(80); //constructor, that sets the bitarray with 80 bits  }  public Packet(byte[] buffer)  {  bitArr = new BitArray(buffer);  Deserialize();  }  #region Deserialization  public void Deserialize()  {  operation = (Operation)GetInt(3);  number1 = GetInt(32);  number2 = GetInt(32);  state = (State)GetInt(2);  sessionId = GetInt(8);  mode = (Mode)GetInt(2);  }  private int GetInt(int length) //deserialize int32 and add to bitarray  {  var result = new int[1];  BitArray tmp = new BitArray(32, false);  for (int i = length-1; i >= 0 ; i--)  {  tmp[i] = getBit();  }  tmp.CopyTo(result, 0);  return result[0];  }  private bool getBit() //getter of last bit in bitarray  {  if (index % 8 == 0)  {  byteIndex++;  index = byteIndex \* 8;  }  index--;  return bitArr[index];  }  #endregion  #region Serialization  public void Serialize()  {  Add(operation);  Add(number1);  Add(number2);  Add(state);  Add(Convert.ToByte(sessionId));  Add(mode);  }  int index = 0;  int byteIndex = 0;  private void Add(bool value) //serialization of bool value  {  if (index % 8 == 0)  {  byteIndex++;  index = byteIndex \* 8;  }  index--;  bitArr.Set(index, value);  }  private void Add(byte value) //serialization of byte value  {  for (int i = 7; i >= 0; i--)  {  this.Add((value & (1 << i)) != 0);  }  }  private void Add(Operation op) //serialization of Operation  {  foreach (byte by in BitConverter.GetBytes((int)op))  {  for (int i = 2; i >= 0; i--)  {  this.Add((by & (1 << i)) != 0);  }  break;  }  }  private void Add(Mode mo) //serialization of mode  {  foreach (byte by in BitConverter.GetBytes((int)mo))  {  for (int i = 1; i >= 0; i--)  {  this.Add((by & (1 << i)) != 0);  }  break;  }  }  private void Add(State st) //serialization of state  {  foreach (byte by in BitConverter.GetBytes((int)st))  {  for (int i = 1; i >= 0; i--)  {  this.Add((by & (1 << i)) != 0);  }  break;  }  }  private void Add(int value) //serialization of int32  {  byte[] binary = BitConverter.GetBytes(value);  foreach (byte by in binary.Reverse())  {  Add(by);  }  }  #endregion  public byte[] GetBytes() //function that returns byte array with serialized bitarray  {  this.Serialize();  byte[] binary = new byte[(bitArr.Length - 1) / 8 + 1];  bitArr.CopyTo(binary, 0);  return binary;  }  }  } |

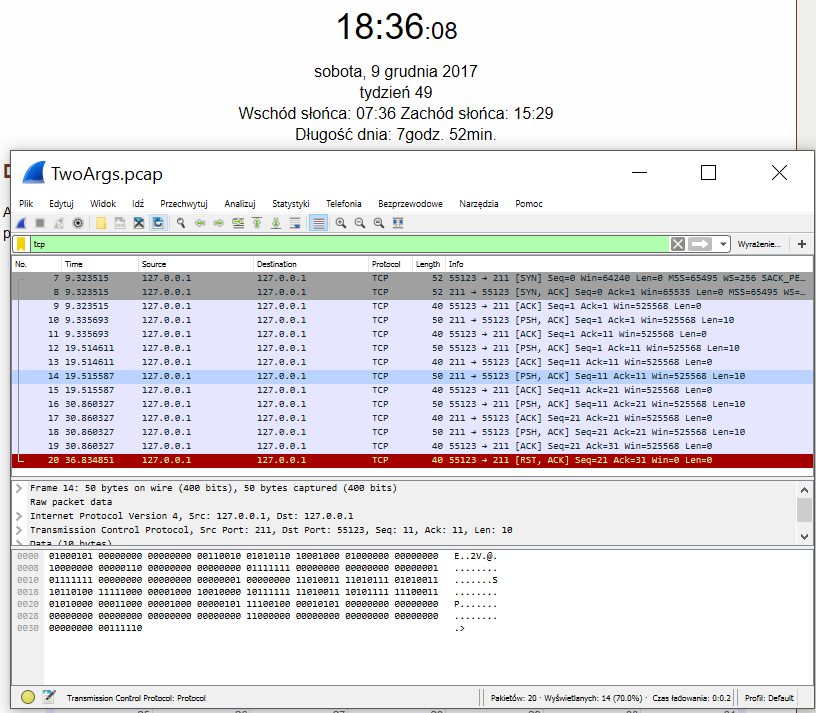
**4. PRZEBIEG PRZYKŁADOWEJ SESJI KOMUNIKACYJNEJ – OPIS SŁOWNY ORAZ OBRAZ SESJI ZAREJESTROWANY PRZEZ PROGRAM WIRESHARK, WRAZ ZE STOSOWNYMI OBJAŚNIENIAMI**

A. Opis słowny

1. Połączenie klienta z serwerem.
2. Uzyskanie ID sesji (wysyłanie stosownego pakietu do klienta)..
3. Wybranie przez klienta sposobu obliczania (2 argumenty lub wiele argumentów).
4. Ustalenie:
   1. liczb 1 i 2 oraz operatora do przesłania (dla 2 argumentowego sposobu)
   2. liczby 1 oraz operatora do przesłania (dla wieloargumentowego sposobu)
5. Serializacja danych.
6. Wysłanie pakietu.
7. Odebranie pakietu przez serwer.
8. Deserialzacja danych.
9. Wykonanie obliczeń na podstawie wybranego sposobu (2 lub wiele argumentów).
10. Przesłanie do klienta pakietu ze zwróconym wynikiem w polu liczby 1.
11. Odebranie pakietu przez serwer.
12. Deserializacja danych.
13. Wyświetlenie wyniku i jeżeli wybrana operacja wieloargumentowa to powrót do kroku 4b.

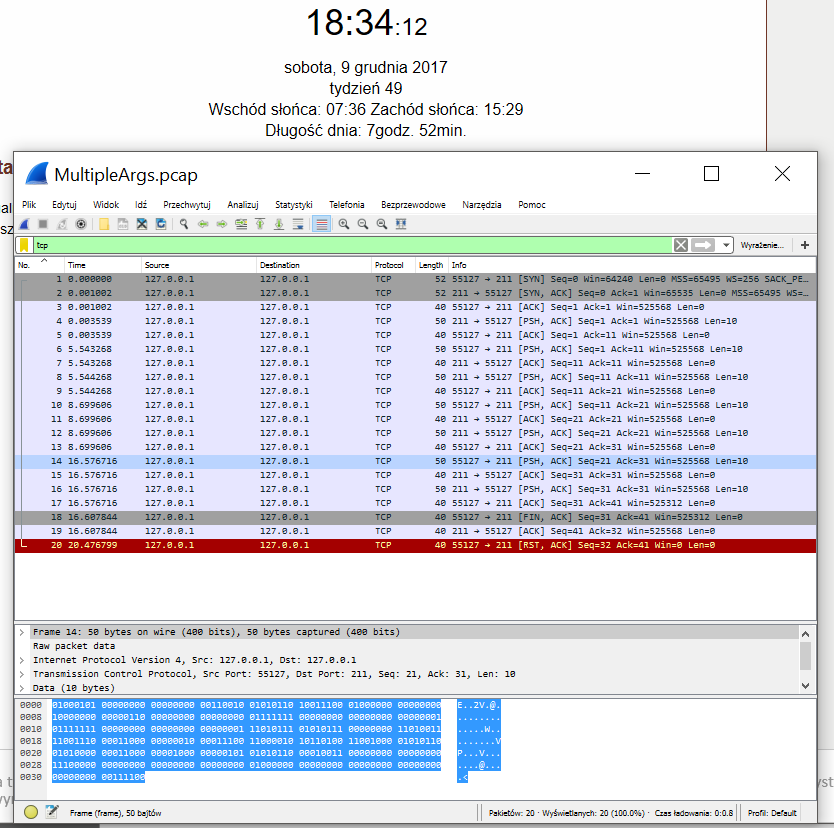
B. Obraz sesji zarejestrowany przez program Wireshark

Dla operacji dwuargumentowej:



Rysunek Transmisja zarejestrowana podczas wykonywania operacji dwuargumentowej.

Dla operacji wieloargumentowej:



Rysunek Transmisja zarejestrowana podczas wykonywania operacji wieloargumentowej.

**5. ODPOWIEDZI NA PYTANIA POSTAWIONE W SEKCJI „ZADANIA SZCZEGÓŁOWE”**

A. Określ teoretyczną oraz rzeczywistą wielkość komunikatów. Czy rozmiar jest zależny od przesyłanych danych? Czy istnieje możliwość łatwej rozbudowy protokołu?

Teoretyczna wielkość pakietu to 79 bitów, a rzeczywista to 80 bitów, czyli 8 bajtów. Rozmiar nie jest zależny od przesyłanych danych, ponieważ zawsze występuje dopełnienie zerami niewykorzystanych bitów. Istnieje łatwa możliwość rozbudowy protokołu, ponieważ wystarczy dodać do klasy Packet.cs odpowiednie pola oraz zmodernizować funkcję serializacji i deserializacji danych.