

Michał Dybaś

Temat projektu:

Projekt i implementacja serwera i klienta programu FTP

Założenia do projektu:

- Zastosować protokół UDP
- Program klienta ma działać na platformach: Linux, Windows.XX,
- Program serwera ma działać na platformach Linux,
- Program serwera ma być serwerem współbieżnym (wątki), –
- Powinna być możliwa zarówno transmisja plików binarnych jak tekstowych.
- Realizacja projektu: java.
- Interfejs - Środowisko tekstowe

Spis treści

1.	Kod źródłowy klasy konfiguracyjnej Config.java	3
2.	Kod źródłowy klienta ftp Client.java.....	4
3.	Kod źródłowy serwera ftp Server.java	21
4.	Wyjaśnienie sposobu rozwiązania podstawowych problemów.....	32
5.	Działanie aplikacji	38

1. Kod źródłowy klasy konfiguracyjnej Config.java

```
package kpu.krosno;

import javax.sound.sampled.Port;
import java.io.IOException;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.Random;
import java.util.Scanner;

public class Config {
    public static final int MAX_HOST = 50; // Maksymalna liczba hostów
    public static final int PORT = 40000; // Znany port serwera
    public static final int BUFFER_SIZE = 1024; // Maksymalny rozmiar
    // bufforu używany podczas przesyłania plików
    public static final int MAX_BUFFER_SIZE = 65507; // Maksymalny rozmiar
    // bufforu, dla przesyłania informacji i komend pomiędzy klientem i serwerem
    public static final int TIMEOUT_MILLISECONDS = 1000; // Czas
    // oczekiwania gniazda UDP na wiadomość zwrotną
    public static final int TIMEOUT_TIMES = 4; // Ilość prób ponownego
    // przesłania wiadomości w przypadku braku odpowiedzi
    public static Scanner scanner = new Scanner(System.in); // Statyczna
    // zmienna dla wejścia klawiatury

    public static String stringFromDatagram(DatagramPacket receiveDatagram)
    // Metoda konwertująca byte z datagramu na String
    {
        if(receiveDatagram == null)
            return null;
        return new String(receiveDatagram.getData(), 0,
            receiveDatagram.getLength(), StandardCharsets.UTF_8);
    }
}
```

2. Kod źródłowy klienta ftp Client.java

```
package kpu.krosno;

import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;

public class Client
{
    public static void main(String[] args)
    {
        // Deklaracja i inicjalizacja zmiennych globalnych
        String state = "welcome";
        String command = "";
        String startPath = Paths.get("").toAbsolutePath().toString();
        String currentPath = startPath;

        DatagramSocket socket = null;
        InetSocketAddress address = null;
        byte[] sendData, receiveData;
        DatagramPacket sendDatagram, receiveDatagram;
        boolean connectionState = false;
        String currentIp = "";

        FileOutputStream fileOutputStream = null;
        File file = null;
        long currentPacket = -1;
        long numberOfPackets = -1;
        long fileSize = -1; // Bytes
        String streamState = "off"; // off, download or upload

        while(true)
        {
            // Wywołanie garbage collector
            System.gc();
            switch(state)
            {
                case "welcome":
                    System.out.print("Witaj w aplikacji, aby zobaczyć listę  
dostępnych komend wpisz help.");
                    state = "options";
                    break;

                case "options":
                    System.out.print("\nWpisz komendę: ");
                    command = Config.scanner.nextLine();

                    if (command.equals("help"))
                        state = "help";
                    else if (command.equals("dir"))
                        state = "dir";
                    else if (command.equals("pwd"))
                        state = "pwd";
                    else if (command.matches("^cd .*"))
                        state = "cd";
            }
        }
    }
}
```

```

        else if (command.matches("^path .*$"))
            state = "path";
        else if (command.matches("^connect .*$"))
            state = "connect";
        else if (command.equals("address"))
            state = "address";
        else if (command.equals("status"))
            state = "status";
        else if (command.equals("sdir"))
            state = "sdir";
        else if (command.matches("^download .*$"))
            state = "download";
        else if (command.matches("^upload .*$"))
            state = "upload";
        else if (command.equals("exit"))
            state = "exit";
        else if (command.equals("help help"))
            System.out.print("Wyświetla listę dostępnych
komend.");
        else if (command.equals("help dir"))
            System.out.print("Wyświetla listę plików i
katalogów w aktualnej ścieżce, a także informacje czy dany plik jest
katalogiem.");
        else if (command.equals("help pwd"))
            System.out.print("Wyświetla ścieżkę aktualnego
katalogu roboczego, do którego będą pobierane pliki z serwera lub z którego
pliki można wysłać na serwer.");
        else if (command.equals("help cd"))
        {
            System.out.println("cd .. -- polecenie to
przechodzi do katalogu nadrzednego jeśli to możliwe");
            System.out.println("cd ~ -- polecenie to przechodzi
do startowego katalogu roboczego");
            System.out.print("cd 'nazwa_katalogu' -- polecenie
to przechodzi do podkatalogu o podanej nazwie.");
        }
        else if (command.equals("help path"))
            System.out.print("path 'nazwa_sciezki' --
przechodzi do podanej sciezki, jesli jest ona katalogiem");
        else if (command.equals("help connect"))
            System.out.print("'adres_ip_serwera' -- próbuje
nawiązać połączenie z serwerem o podanym adresie ip");
        else if (command.equals("help address"))
            System.out.print("Wyświetla informację na temat
aktualnego adresu ip serwera.");
        else if (command.equals("help status"))
            System.out.print("Wyświetla informację na temat
aktualnego statusu połączenia z serwerem.");
        else if (command.equals("help sdir"))
            System.out.print("Pobiera z serwera listę katalogów
i plików w aktualnym folderze roboczym.");
        else if (command.equals("help download"))
            System.out.print("download 'nazwa_pliku' -- pobiera
podany plik z serwera do aktualnego folderu roboczego. !Uwaga, jeśli plik o
takiej nazwie już istnieje, zostanie on nadpisany.");
        else if (command.equals("help upload"))
            System.out.print("upload 'nazwa_pliku' -- wysyła
podany plik z katalogu roboczego na serwer. !Uwaga, jeśli plik o takiej
nazwie już istnieje, zostanie on nadpisany.");
        else if (command.equals("help exit"))
            System.out.println("Kończy pracę programu, jeśli

```

```

aplikacja jest połączona z serwerem, polecenie kończy uprzednio połączenie
z serwerem.");
        else
            System.out.print("Komenda '" + command + "' jest
nie prawidłowa, aby zobaczyć listę dostępnych komend wpisz help.");
            break;

        case "dir":
            File directory = new File(currentPath);
            File files[] = directory.listFiles();

            for(int i = 0; i < files.length; i++)
            {
                System.out.print(files[i].getName());
                if(files[i].isDirectory())
                    System.out.print("\tkatalog");
                System.out.print("\t" + files[i].length() +
"B");

                if(!files[i].canRead())
                    System.out.print("\tCan't read");
                if(i != files.length - 1)
                    System.out.print("\n");
            }
            state = "options";
            break;

        case "pwd":
            System.out.print("Aktualna ścieżka katalogu roboczego:
" + currentPath);
            state = "options";
            break;

        case "cd":
            if(command.equals("cd .."))
            {
                String dirs[] = currentPath.split("\\\\");
                if(dirs.length <= 1)
                    System.out.print("Nie można przejść do wyższego
katalogu.");

                else {
                    currentPath = "";
                    for(int i = 0; i < dirs.length - 1; i++)
                    {
                        currentPath = Path.of(currentPath,
dirs[i]).toString();
                    }
                    System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
                }
            }
            else if(command.equals("cd ~"))
            {
                currentPath = startPath;
                System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
            }
            else {
                String dirName =
command.substring(command.indexOf(' ') + 1);
                if (!dirName.equals(".") && new
File(Path.of(currentPath, dirName).toString()).isDirectory())

```

```

        {
            currentPath = Path.of(currentPath,
dirName).toString();
            System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
        }
        else
        {
            System.out.println("Katalog o nazwie '" +
dirName + "', nie istnieje lub nie jest katalogiem.");
            System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
        }
    }
    state = "options";
    break;

    case "path":
        String newPath = command.substring(command.indexOf(' ')
+ 1);
        if (new File(Path.of(newPath).toString()).exists() &&
new File(Path.of(newPath).toString()).isDirectory())
        {
            currentPath = Path.of(newPath).toString();
            System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
        }
        else
        {
            System.out.println("Ścieżka '" + newPath + "', nie
istnieje lub nie jest katalogiem.");
            System.out.print("Aktualna ścieżka katalogu
roboczego: " + currentPath);
        }
        state = "options";
        break;

    case "connect":
    {
        if (socket == null) {
            try {
                socket = new DatagramSocket();

socket.setSoTimeout(Config.TIMEOUT_MILLISECONDS);
            }
            catch (SocketException exception)
            {
                System.out.print("Nie udało utworzyć się
gniazda klienta dla połączenia z serwerem.");
                connectionState = false;
                state = "options";
                break;
            }
        }
        currentIp = command.substring(command.indexOf(' ') +
1);

        try
        {
            address = new
InetSocketAddress(InetAddress.getByName(currentIp), Config.PORT);
        }
    }

```

```

        catch (UnknownHostException exception)
        {
            System.out.print("Podano nieprawidłowy adres ip
serwera.");

            address = null;
            connectionState = false;
            state = "options";
            break;
        }
        String msg = "CONNECT";
        sendData = msg.getBytes(StandardCharsets.UTF_8);
        sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
        receiveData = new byte[Config.MAX_BUFFER_SIZE];
        receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);

        boolean errorFlag = true;
        for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
        {
            try {
                socket.send(sendDatagram);
                socket.receive(receiveDatagram);
                errorFlag = false;
                break;
            }
            catch (IOException exception) {
            }
        }
        if(errorFlag)
        {
            System.out.print("Nie udało nawiązać się połączenia
z serwerem.");

            connectionState = false;
        }
        else
        {
            msg = Config.stringFromDatagram(receiveDatagram);
            if(msg.equals("SERVER_FULL"))
            {
                System.out.print("Nie udało nawiązać się
połączenia z serwerem. - Serwer jest przepełniony.");
                connectionState = false;
            }
            elseif(msg.equals("ALREADY_CONNECTED")) {
                System.out.print("Powiązanie z tym serwerem już
zostało nawiązane.");
                connectionState = true;
            }
            else if(msg.equals("CONNECTED"))
            {
                System.out.print("Powiązanie z serwerem zostało
nawiązane.");
                connectionState = true;
            }
        }
    }
    state = "options";
    break;

    case "address":

```



```

        if(currentIp == null || currentIp.isEmpty())
            System.out.print("Nie podano adresu ip serwera.");
        else {
            try
            {
                System.out.print(InetAddress.getByName(currentIp).getHostAddress());
            }
            catch (UnknownHostException exception)
            {
                System.out.println("Podany adres ip serwera
jest nieprawidłowy.");
            }
            finally
            {
                System.out.print("Adres ip serwera: " +
currentIp + ", port serwera: " + Config.PORT);
            }
        }
        state = "options";
        break;

        case "status":
            if(!connectionState)
            {
                System.out.print("Aktualnie nie jesteś połączony z
żadnym serwerem.");
            }
            else
            {
                String msg = "STATUS";
                sendData = msg.getBytes(StandardCharsets.UTF_8);
                sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
                receiveData = new byte[Config.MAX_BUFFER_SIZE];
                receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);

                boolean errorFlag = true;
                for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
                {
                    try {
                        socket.send(sendDatagram);
                        socket.receive(receiveDatagram);
                        errorFlag = false;
                        break;
                    }
                    catch (IOException exception) {
                    }
                }
                if(errorFlag)
                {
                    System.out.print("Aktualnie nie jesteś
połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
                    connectionState = false;
                }
                else
                {
                    msg =
Config.stringFromDatagram(receiveDatagram);
                    if(msg.equals("CONNECTED"))

```

```

        System.out.print("Aktualnie jesteś
połączony z serwerem o adresie: " + currentIp + ":" + Config.PORT);
    else if(msg.equals("NOT_CONNECTED"))
    {
        System.out.print("Aktualnie nie jesteś
połączony z żadnym serwerem.");
        connectionState = false;
    }
    }
    state = "options";
    break;

    case "sdir":
        if(!connectionState)
        {
            System.out.print("Aktualnie nie jesteś połączony z
żadnym serwerem.");
        }
        else
        {
            String msg = "SDIR";
            sendData = msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
            receiveData = new byte[Config.MAX_BUFFER_SIZE];
            receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);

            boolean errorFlag = true;
            for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
            {
                try {
                    socket.send(sendDatagram);
                    socket.receive(receiveDatagram);
                    errorFlag = false;
                    break;
                }
                catch (IOException exception) {
                }
            }
            if(errorFlag)
            {
                System.out.print("Aktualnie nie jesteś
połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
                connectionState = false;
            }
            else
            {
                msg =
Config.stringFromDatagram(receiveDatagram);
                System.out.print(msg);
            }
        }
        state = "options";
        break;

    case "download":
        if(!connectionState)
        {
            System.out.print("Aktualnie nie jesteś połączony z

```

```

żadnym serwerem.");
    }
    else
    {
        String msg = "DOWNLOAD " +
command.substring(command.indexOf(' ') + 1);
        sendData = msg.getBytes(StandardCharsets.UTF_8);
        sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
        receiveData = new byte[Config.MAX_BUFFER_SIZE];
        receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);

        boolean errorFlag = true;
        for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
        {
            try {
                socket.send(sendDatagram);
                socket.receive(receiveDatagram);
                errorFlag = false;
                break;
            }
            catch (IOException exception) {
            }
        }
        if(errorFlag)
        {
            System.out.print("Aktualnie nie jesteś
połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
            connectionState = false;
        }
        else
        {
            msg =
Config.stringFromDatagram(receiveDatagram);
            if(msg.equals("NOT_EXIST"))
                System.out.print("Podany plik nie
istnieje.");
            else if(msg.equals("CAN'T_READ"))
                System.out.print("Brak uprawnień do odczytu
podanego pliku.");
            else if(msg.equals("DIRECTORY"))
                System.out.print("Podany plik jest
katalogiem. Można pobierać tylko pojedyncze pliki.");
            else if(msg.matches("[0-9]+ [0-9]+ [0-9]+$"))
            {
                file = new File(Path.of(currentPath,
command.substring(command.indexOf(' ') + 1)).toString());
                fileSize = -1; // Bytes
                currentPacket = -1;
                numberOfPackets = -1;
                streamState = "off";

                try
                {
                    file.delete();
                    file.createNewFile();
                    if(!file.canWrite())
                        throw new IOException();
                    fileOutputStream = new
FileOutputStream(file);

```

```

    }
    catch (IOException exception)
    {
        try
        {
            msg = "-1";
            sendData =
msg.getBytes(StandardCharsets.UTF_8);
            sendDataGram = new
DatagramPacket(sendData, sendData.length, address);
            socket.send(sendDatagram);
            socket.receive(receiveDatagram);
            if(fileOutputStream != null)
                fileOutputStream.close();
            fileOutputStream = null;
            errorFlag = false;
            break;
        }
        catch (IOException subexception) { }
        System.out.print("Nie można utworzyć
takiego pliku -- Odmowa dostępu.");
        state = "options";
        break;
    }

    String tokens[] = msg.split(" ");
    currentPacket = Long.parseLong(tokens[0]);
    numberOfPackets =
Long.parseLong(tokens[1]);
    fileSize = Long.parseLong(tokens[2]); //
Bytes
    streamState = "download";
    System.out.println("Pobieranie pliku o
nazwie: " + file.getName());

    while (true)
    {
        msg = currentPacket + " " +
numberOfPackets + " " + fileSize;
        sendData =
msg.getBytes(StandardCharsets.UTF_8);
        sendDataGram = new
DatagramPacket(sendData, sendData.length, address);
        receiveData = new
byte[Config.BUFFER_SIZE];
        receiveDatagram = new
DatagramPacket(receiveData, receiveData.length);

        errorFlag = true;
        for (int i = 0; i <
Config.TIMEOUT_TIMES * 2; i++)
        {
            try
            {
                socket.send(sendDatagram);

socket.receive(receiveDatagram);

                errorFlag = false;
                break;
            } catch (IOException exception) { }
        }
    }

```

```

        if (errorFlag)
        {
            if(file != null)
                file.delete();
            file = null;
            fileSize = -1; // Bytes
            currentPacket = -1;
            numberOfPackets = -1;
            streamState = "off";
            System.out.print("Aktualnie nie
jesteś połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
            connectionState = false;
            try
            {
                if(fileOutputStream != null)
                    fileOutputStream.close();
                fileOutputStream = null;
            }
            catch (IOException ignored) {}
            break;
        }
        else
        {
            byte[] data =

receiveDatagram.getData();

            int len = Config.BUFFER_SIZE;
            if(currentPacket + 1 ==

numberOfPackets)
            {
                len = (int) (fileSize -
(numberOfPackets - 1) * Config.BUFFER_SIZE);
                data = Arrays.copyOfRange(data,
0, len);
            }

            if(len !=

receiveDatagram.getLength())

                continue;

            try
            {
                fileOutputStream.write(data);
            }
            catch (IOException e) {
                try
                {
                    msg = "-1";
                    sendData =

msg.getBytes(StandardCharsets.UTF_8);

                    sendDatagram = new
DatagramPacket(sendData, sendData.length, address);
                    socket.send(sendDatagram);

                    socket.receive(receiveDatagram);

                    errorFlag = false;
                    break;
                }
                catch (IOException

```

```

subexception) { }

        if (file != null)
            file.delete();
        file = null;
        fileSize = -1; // Bytes
        currentPacket = -1;
        numberOfPackets = -1;
        streamState = "off";
        System.out.print("Wystąpił błąd
podczas zapisu pobieranego pliku.");

        break;
    }

    currentPacket++;
    if(currentPacket ==
numberOfPackets)
    {
        file = null;
        fileSize = -1; // Bytes
        currentPacket = -1;
        numberOfPackets = -1;
        streamState = "off";
        try
        {
            msg = "0";
            sendData =

            sendData = new
            DatagramPacket(sendData, sendData.length, address);
            socket.send(sendDatagram);

            socket.receive(receiveDatagram);

            errorFlag = false;
            if(fileOutputStream !=
null)

            fileOutputStream.close();

            fileOutputStream = null;
            break;
        }
        catch (IOException

        System.out.print("Pobieranie
pliku zakończyło się pomyślnie.");

        break;
    }
}
}
}
}
}
}
}
state = "options";
break;

case "upload":
    if(!connectionState)
    {
        System.out.print("Aktualnie nie jesteś połączony z
żadnym serwerem.");
    }
    else

```

```

        {
            String fileName =
command.substring(command.indexOf(' ') + 1);
            file = new File(Path.of(currentPath,
fileName).toString());
            if(!file.exists())
            {
                System.out.print("Nie można wysłać pliku: " +
fileName + " -- podany plik nie istnieje.");
                file = null;
                state = "options";
                break;
            }
            else if(file.isDirectory())
            {
                System.out.print("Nie można wysłać pliku: " +
fileName + " -- podany plik jest katalogiem.");
                file = null;
                state = "options";
                break;
            }
            else if(!file.canRead())
            {
                System.out.print("Nie można wysłać pliku: " +
fileName + " -- odmowa dostępu.");
                file = null;
                state = "options";
                break;
            }
        }

        String msg = "UPLOAD " + fileName;
        sendData = msg.getBytes(StandardCharsets.UTF_8);
        sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
        receiveData = new byte[Config.MAX_BUFFER_SIZE];
        receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);
        byte[] fileData = new byte[Config.BUFFER_SIZE];

        boolean errorFlag = true;
        for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
        {
            try {
                socket.send(sendDatagram);
                socket.receive(receiveDatagram);
                errorFlag = false;
                break;
            }
            catch (IOException exception) {
            }
        }
        if(errorFlag)
        {
            System.out.print("Aktualnie nie jesteś
połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
            file = null;
            connectionState = false;
        }
        else
        {
            msg =

```

```

Config.stringFromDatagram(receiveDatagram);
        if(msg.equals("CAN'T_WRITE")) {
            System.out.print("Brak uprawnień do odczytu
podanego pliku.");
            file = null;
        }
        else if(msg.equals("WAITING")) {
            currentPacket = -1;
            numberOfPackets = (long)
Math.ceil(file.length() / (double)Config.BUFFER_SIZE);
            fileSize = file.length();
            streamState = "upload";
            msg = "0 " + numberOfPackets + " " +
fileSize;

            sendDatagram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
receiveDatagram.getSocketAddress());
            FileInputStream fileInputStream = null;
            try {
                fileInputStream = new
FileInputStream(file);

                socket.send(sendDatagram);
                System.out.println("Wysłanie informacji
do serwera o rozmiarze wysyłanego pliku: " + fileName);
                System.out.println("Rozpoczęcie
wysyłania pliku o nazwie: " + fileName);
            }
            catch (IOException exception)
            {
                currentPacket = -1;
                numberOfPackets = -1;
                fileSize = -1;
                streamState = "off";
                try {
                    fileInputStream.close();
                } catch (IOException ignored) {}
                file = null;
                System.out.print("Wystąpił błąd podczas
wysyłania pliku: " + fileName);
                break;
            }
            while(true)
            {
                errorFlag = true;
                for(int i = 0; i <
Config.TIMEOUT_TIMES; i++)
                {
                    try {
                        socket.send(sendDatagram);

                        socket.receive(receiveDatagram);

                        errorFlag = false;
                        break;
                    }
                    catch (IOException exception) {
                    }
                }
                if(errorFlag)
                {
                    currentPacket = -1;

```



```

        numberOfPackets = -1;
        fileSize = -1;
        streamState = "off";
        try {
            fileInputStream.close();
        } catch (IOException ignored) {}
        file = null;
        connectionState = false;
        System.out.print("Aktualnie nie
jesteś połączony z żadnym serwerem. -- brak odpowiedzi od serwera.");
        break;
    }
    else
    {
        msg =
Config.stringFromDatagram(receiveDatagram);
        if (msg.matches("[0-9]+ [0-9]+ [0-9]+"))
        {
            try
            {
                String tokens[] =
msg.split(" ");
                Long.parseLong(tokens[0]);
                Long.parseLong(tokens[1]);
                Long.parseLong(tokens[2]);
                Config.BUFFER_SIZE;

                if (receiveCurrentPacket +
1 == receiveNumberOfPackets)
                {
                    len = (int)
(receiveFileSize - (receiveNumberOfPackets - 1) * Config.BUFFER_SIZE);
                    if (currentPacket == -1) {
                        fileData = new
byte[len];
                        fileInputStream.read(fileData, 0, len);

                        currentPacket = 0;
                    }
                    if (currentPacket <
receiveCurrentPacket) {
                        fileData = new
byte[len];
                        fileInputStream.read(fileData, 0, len);

                        currentPacket =
receiveCurrentPacket;
                    }

                    sendDatagram = new
DatagramPacket(fileData, fileData.length,
receiveDatagram.getSocketAddress());
                }
            }
        }
    }
}

```

```

        catch (IOException exception)
        {
            currentPacket = -1;
            numberOfPackets = -1;
            fileSize = -1;
            streamState = "off";
            try {

                } catch (IOException

            file = null;
            System.out.print("Błąd

            break;

        }
    }
    else if(msg.equals("0") ||

    {
        if(msg.equals("0"))
            System.out.print("Plik: " +

        else if(msg.equals("-1"))
            System.out.print("Plik: " +

        fileName + " nie został wysłany poprawnie - błąd po stronie serwera.");
        try
        {
            fileInputStream.close();
            fileInputStream = null;
        }
        catch (IOException ignored) {}
        file = null;
        currentPacket = -1;
        numberOfPackets = -1;
        fileSize = -1;
        streamState = "off";
        break;
    }
    else if(msg.equals("WAITING"))
    {
        currentPacket = -1;
        numberOfPackets = (long)

        Math.ceil(file.length() / (double) Config.BUFFER_SIZE);
        fileSize = file.length();
        streamState = "upload";
        msg = "0 " + numberOfPackets +

        " " + fileSize;

        sendDatagram = new
        DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
        msg.getBytes(StandardCharsets.UTF_8).length,
        receiveDatagram.getSocketAddress());

        fileInputStream = null;
        try {
            fileInputStream = new

            socket.send(sendDatagram);

        System.out.println("Wysłanie informacji do serwera o rozmiarze wysyłanego
        pliku: " + fileName);

```

```

System.out.println("Rozpoczęcie wysyłania pliku o nazwie: " + fileName);
    } catch (IOException exception)
    {
        currentPacket = -1;
        numberOfPackets = -1;
        fileSize = -1;
        streamState = "off";
        try {

fileInputStream.close();

        } catch (IOException
ignored) {
        }
        file = null;
        System.out.print("Wystąpił
błąd podczas wysyłania pliku: " + fileName);
        break;
    }
    }
    }
    }
    }
    }
    state = "options";
    break;

    case "exit":
        if(connectionState)
        {
            String msg = "DISCONNECT";
            sendData = msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new DatagramPacket(sendData,
sendData.length, address);
            try
            {
                socket.send(sendDatagram);
            }
            catch (IOException exception) {
            }
            System.out.println("Połączenie z serwerem zostało
zakończone.");
        }
        System.out.println("Aplikacja zostanie wyłączona.");
        System.exit(0);
        break;

    case "help":
        state = "options";
        System.out.println("Dostępne komendy:");
        System.out.println("help -- wyświetla listę komend");
        System.out.println("help 'nazwa_komendy' -- wyświetla
informację na temat danej komendy");
        System.out.println("dir -- wyświetla listę plików i
katalogów w aktualnej ścieżce");
        System.out.println("pwd -- wyświetla aktualną
ścieżkę");
        System.out.println("cd -- przechodzi lub wychodzi z
katalogu");
        System.out.println("path 'nowa_sciezka' -- przechodzi
do podanej ścieżki");

```

```
        System.out.println("connect 'adres_ip_serwera' --  
nawiązuje połączenie z serwerem o podanym adresie");  
        System.out.println("address -- wyświetla informacje na  
temat aktualnego adresu ip serwera");  
        System.out.println("status -- wyświetla informacje na  
temat aktualnego połączenia z serwerem");  
        System.out.println("sdir -- wyświetla listę plików  
możliwych do pobrania z serwera");  
        System.out.println("download 'nazwa_pliku' -- pobiera  
plik z serwera do aktualnego katalogu roboczego");  
        System.out.println("upload 'nazwa_pliku' -- wysyła plik  
z aktualnego katalogu roboczego na serwer");  
        System.out.print("exit -- wychodzi z programu");  
        break;  
  
        default:  
            state = "options";  
            System.out.print("\nKomenda '" + command + "' jest nie  
prawidłowa, aby zobaczyć listę dostępnych komend wpisz help.");  
            break;  
    }  
}  
  
}
```

3. Kod źródłowy serwera ftp Server.java

```
package kpu.krosno;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;

class Connection {
    public String connectionName;
    public String startPath;
    public String currentPath = "";

    public FileInputStream threadFileInputStream = null;
    public File file = null;
    public byte[] fileData = null;
    public long fileSize = -1; // Bytes
    public long currentPacket = -1;
    public long numberOfPackets = -1;
    public String streamState = "off"; // off, download or upload

    public Connection(String connectionName, String startPath) {
        this.connectionName = connectionName;
        this.startPath = startPath;
    }

    public void disconnect()
    {
        if(this.threadFileInputStream != null)
        {
            try {
                this.threadFileInputStream.close();
            }
            catch (IOException exception) {}
            this.threadFileInputStream = null;
        }
        if(this.file != null && this.streamState.equals("upload") &&
this.currentPacket + 1 < this.numberOfPackets)
            this.file.delete();
        this.file = null;
        System.out.println("SERWER-MSG(THREAD:" + this.connectionName + "):
Zakończono połączenie z klientem.");
    }
}

class ServerThread implements Runnable
{
    Thread thread;
    String threadName;
```

```

Connection threadConnection;
ArrayList<Connection> connections;
DatagramSocket threadSocket;
DatagramPacket receiveDatagram;
String startPath;
boolean foundFlag;
String lastMsg;

ServerThread (String threadName, ArrayList<Connection> connections,
DatagramPacket receiveDatagram, String startPath) throws SocketException
{
    this.threadName = threadName;
    this.connections = connections;
    this.receiveDatagram = receiveDatagram;
    this.threadSocket = new DatagramSocket();
    this.startPath = startPath;
    this.foundFlag = false;
    this.lastMsg = Config.stringFromDatagram(receiveDatagram);
    this.thread = new Thread(this, threadName);
    this.thread.start();
}

public void run()
{
    if(!connections.isEmpty())
    {
        for(int i = 0; i < connections.size(); i++)
        {
            if(this.threadName.equals(connections.get(i).connectionName))
            {
                this.threadConnection = connections.get(i);
                this.foundFlag = true;
                break;
            }
        }
    }

    if(!this.foundFlag && connections.size() > Config.MAX_HOST)
    {
        String msg = "SERVER_FULL";
        DatagramPacket sendDatagram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
this.receiveDatagram.getSocketAddress());
        try
        {
            this.threadSocket.send(sendDatagram);
            System.out.println("SERWER-MSG(THREAD:" + this.threadName +
"): Odrzucono połączenie z klientem z powodu braku miejsca w kolejce.");
        } catch (IOException ignored) {}
        this.threadSocket.close();
    }

    if(this.lastMsg.equals("CONNECT"))
    {
        String msg = "ALREADY_CONNECTED";
        if(!this.foundFlag)
        {
            msg = "CONNECTED";
            this.threadConnection = new Connection(this.threadName,

```

```

startPath);
        connections.add(this.threadConnection);
        this.foundFlag = true;
    }
    DatagramPacket sendDatagram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
this.receiveDatagram.getSocketAddress());
    try
    {
        this.threadSocket.send(sendDatagram);
        System.out.println("SERWER-MSG(THREAD:" + this.threadName +
"): Nawiazano nowe połączenie z klientem.");
    } catch (IOException ignored) {}
    this.threadSocket.close();
}
if(this.lastMsg.equals("DISCONNECT"))
{
    if(this.foundFlag)
    {
        this.threadConnection.disconnect();
        this.connections.remove(this.threadConnection);
    }
    this.threadSocket.close();
}
else if(this.lastMsg.equals("STATUS"))
{
    String msg = "NOT_CONNECTED";
    if(this.foundFlag)
    {
        msg = "CONNECTED";
    }
    DatagramPacket sendDatagram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
this.receiveDatagram.getSocketAddress());
    try
    {
        this.threadSocket.send(sendDatagram);
        System.out.println("SERWER-MSG(THREAD:" + this.threadName +
"): Wysłano do klienta informację na temat statusu połączenia.");
    } catch (IOException ignored) {}
    this.threadSocket.close();
}
else if(this.lastMsg.equals("SDIR"))
{
    if (this.foundFlag)
    {
        File dirname = new
File(Path.of(this.threadConnection.startPath,
this.threadConnection.currentPath).toString());
        File files[] = dirname.listFiles();
        StringBuilder stringBuilder = new StringBuilder();
        if(files.length < 1)
            stringBuilder.append("Brak plików.");
        else
        {
            for(int i = 0; i < files.length; i++)
            {
                stringBuilder.append(files[i].getName());
                if(files[i].isDirectory())

```

```

        stringBuilder.append("\tkatalog");
        stringBuilder.append("\t" + files[i].length() +
"B");

        if(!files[i].canRead())
            stringBuilder.append("\tCan't read");
        if(i != files.length - 1)
            stringBuilder.append("\n");
    }
    String msg = stringBuilder.toString();
    DatagramPacket sendDataGram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
this.receiveDatagram.getSocketAddress());
    try
    {
        this.threadSocket.send(sendDatagram);
        System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Wysłano listę plików do klienta.");
    } catch (IOException ignored) {}
    this.threadSocket.close();
}

}
else if(this.lastMsg.matches("DOWNLOAD .*"))
{
    if(this.foundFlag)
    {
        String fileName =
Config.stringFromDatagram(this.receiveDatagram);
        fileName = fileName.substring(fileName.indexOf(' ') + 1);
        this.threadConnection.file = new
File(Path.of(Path.of(this.threadConnection.startPath,
this.threadConnection.currentPath).toString(), fileName).toString());

        String msg = "";
        if(!this.threadConnection.file.exists())
        {
            this.threadConnection.file = null;
            this.threadConnection.currentPacket = -1;
            this.threadConnection.numberOfPackets = -1;
            this.threadConnection.fileSize = -1;
            this.threadConnection.streamState = "off";
            msg = "NOT_EXIST";
            System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Odmowa wysłania pliku(Plik nie istnieje): " +
fileName);
        }
        else if(this.threadConnection.file.isDirectory())
        {
            this.threadConnection.file = null;
            this.threadConnection.currentPacket = -1;
            this.threadConnection.numberOfPackets = -1;
            this.threadConnection.fileSize = -1;
            this.threadConnection.streamState = "off";
            msg = "DIRECTORY";
            System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Odmowa wysłania pliku(Plik jest katalogiem): " +
fileName);
        }
        else if(!this.threadConnection.file.canRead())
        {

```



```

        this.threadConnection.file = null;
        this.threadConnection.currentPacket = -1;
        this.threadConnection.numberOfPackets = -1;
        this.threadConnection.fileSize = -1;
        this.threadConnection.streamState = "off";
        msg = "CAN'T_READ";
        System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Odmowa wysłania pliku(Brak uprawnień odczytu pliku
przez serwer): " + fileName);
    }
    else
    {
        this.threadConnection.currentPacket = -1;
        this.threadConnection.numberOfPackets = (long)
Math.ceil(this.threadConnection.file.length() /
(double)Config.BUFFER_SIZE);
        this.threadConnection.fileSize =
this.threadConnection.file.length();
        this.threadConnection.streamState = "download";
        msg = "0 " + this.threadConnection.numberOfPackets + "
" + this.threadConnection.fileSize;
    }
    DatagramPacket sendDataGram = new
DatagramPacket(msg.getBytes(StandardCharsets.UTF_8),
msg.getBytes(StandardCharsets.UTF_8).length,
this.receiveDatagram.getSocketAddress());
    try
    {
        this.threadSocket.send(sendDatagram);
        this.threadConnection.threadFileInputStream = new
FileInputStream(this.threadConnection.file);
        System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Wysłanie informacji do klienta o rozmiarze wysyłanego
pliku: " + fileName);
        System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Rozpoczęcie wysyłania pliku o nazwie: " + fileName);
    } catch (IOException ignored) {}
    this.threadSocket.close();
}
else if(this.lastMsg.equals("0") || this.lastMsg.equals("-1"))
{
    if(this.foundFlag)
    {
        if(this.threadConnection.streamState == "download")
        {
            if(this.lastMsg.equals("0"))
                System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Plik: " + this.threadConnection.file.getName() + "
został wysłany poprawnie.");
            else if(this.lastMsg.equals("-1"))
                System.out.println("SERWER-MSG(THREAD:" +
this.threadName + "): Plik: " + this.threadConnection.file.getName() + "
nie został wysłany poprawnie, błąd po stronie klienta.");
            try
            {
this.threadConnection.threadFileInputStream.close();
                this.threadConnection.threadFileInputStream = null;
            }
            catch (IOException ignored) {}

```

```

        this.threadConnection.file = null;
        this.threadConnection.currentPacket = -1;
        this.threadConnection.numberOfPackets = -1;
        this.threadConnection.fileSize = -1;
        this.threadConnection.streamState = "off";
    }
}
else if(this.lastMsg.matches("[0-9]+ [0-9]+ [0-9]+$"))
{
    if(this.foundFlag)
    {
        try
        {
            String tokens[] = lastMsg.split(" ");
            long currentPacket = Long.parseLong(tokens[0]);
            long numberOfPackets = Long.parseLong(tokens[1]);
            long fileSize = Long.parseLong(tokens[2]);
            int len = Config.BUFFER_SIZE;

            if(currentPacket + 1 == numberOfPackets)
            {
                len = (int) (fileSize - (numberOfPackets - 1) *
Config.BUFFER_SIZE);
            }

            if(this.threadConnection.currentPacket == -1)
            {
                this.threadConnection.fileData = new byte[len];

this.threadConnection.threadFileInputStream.read(this.threadConnection.file
Data, 0 , len);

                this.threadConnection.currentPacket = 0;
            }

            if(currentPacket > this.threadConnection.currentPacket)
            {
                this.threadConnection.fileData = new byte[len];

this.threadConnection.threadFileInputStream.read(this.threadConnection.file
Data, 0 , len);

                this.threadConnection.currentPacket =
currentPacket;
            }

            DatagramPacket sendDatagram = new
DatagramPacket(this.threadConnection.fileData,
this.threadConnection.fileData.length,
this.receiveDatagram.getSocketAddress());
            this.threadSocket.send(sendDatagram);
        }
        catch (IOException ignored)
        { }
        this.threadSocket.close();
    }
}
else if(this.lastMsg.matches("UPLOAD .*"))
{
    if(this.foundFlag)
    {
        this.threadConnection.streamState = "off";
    }
}

```

```
String fileName =
Config.stringFromDatagram(this.receiveDatagram);
    fileName = fileName.substring(fileName.indexOf(' ') + 1);
    this.threadConnection.file = new
File(Path.of(Path.of(this.threadConnection.startPath,
this.threadConnection.currentPath).toString(), fileName).toString());
    String msg = "";
    FileOutputStream fileOutputStream = null;
    DatagramPacket sendDatagram = null;
    byte[] sendData = new byte[Config.MAX_BUFFER_SIZE];
    boolean errorFlag = true;
    try
    {
        this.threadConnection.file.delete();
        this.threadConnection.file.createNewFile();
        if(!this.threadConnection.file.canWrite())
            throw new IOException();
        fileOutputStream = new
FileOutputStream(this.threadConnection.file);
        errorFlag = false;
    }
    catch (IOException exception)
    {
        try
        {
            msg = "CAN'T WRITE";
            sendData = msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new DatagramPacket(sendData,
sendData.length, this.receiveDatagram.getSocketAddress());
            this.threadSocket.send(sendDatagram);
            if(fileOutputStream != null)
                fileOutputStream.close();
            fileOutputStream = null;
        }
        catch (IOException subexception) { }
        System.out.println("SERWER-MSG (THREAD:" +
this.threadName + "): Nie można pobrać pliku: " + fileName + " od klienta -
- odmowa dostępu.");
    }
    if(!errorFlag)
    {
        try
        {
            msg = msg = "WAITING";
            sendData = msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new DatagramPacket(sendData,
sendData.length, this.receiveDatagram.getSocketAddress());
            this.threadSocket.send(sendDatagram);

this.threadSocket.connect(this.receiveDatagram.getSocketAddress());

            if(this.threadConnection.streamState.equals("off"))
            {
                errorFlag = true;
                for (int i = 0; i < Config.TIMEOUT_TIMES; i++)
                {
                    try
                    {

```

```

Config.stringFromDatagram(this.receiveDatagram);
        if (msg.matches("[0-9]+ [0-9]+ [0-9]+"))
        {
            String tokens[] = msg.split(" ");
            this.threadConnection.currentPacket
= 0;

            this.threadConnection.numberOfPackets = Long.parseLong(tokens[1]);
            this.threadConnection.fileSize =
Long.parseLong(tokens[2]); // Bytes
            this.threadConnection.streamState =
"upload";

            System.out.println("SERVER-
MSG(THREAD:" + this.threadName + "): Rozpoczęcie pobierania pliku o nazwie:
" + fileName);

            errorFlag = false;
            break;
        }
        this.threadSocket.send(sendDatagram);
    }
    catch (IOException exception) { }
    }

    if(!errorFlag)
        while (true)
        {
            msg = this.threadConnection.currentPacket +
" " + this.threadConnection.numberOfPackets + " " +
this.threadConnection.fileSize;
            sendData =
msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new DatagramPacket(sendData,
sendData.length);

            byte[] receiveData = new
byte[Config.BUFFER_SIZE];
            this.receiveDatagram = new
DatagramPacket(receiveData, receiveData.length);

            errorFlag = true;
            for (int i = 0; i < Config.TIMEOUT_TIMES *
2; i++)
            {
                try
                {
                    this.threadSocket.send(sendDatagram);

                    this.threadSocket.receive(this.receiveDatagram);
                    errorFlag = false;
                    break;
                } catch (IOException exception) { }
            }
            if (errorFlag)
            {
                if(this.threadConnection.file != null)
this.threadConnection.file.delete();

                this.threadConnection.file = null;
                this.threadConnection.fileSize = -1; //

```

```

Bytes
    this.threadConnection.currentPacket = -
1;
    this.threadConnection.numberOfPackets =
-1;
    this.threadConnection.streamState =
"off";
    System.out.println("SERWER-MSG (THREAD:"
+ this.threadName + "): Wystąpił błąd podczas pobierania pliku: " +
fileName + " -- klient nie odpowiada.");
    try
    {
        if(fileOutputStream != null)
            fileOutputStream.close();
        fileOutputStream = null;
    }
    catch (IOException ignored) {}
    break;
}
else
{
    int len = Config.BUFFER_SIZE;
    byte[] data =
this.receiveDatagram.getData();
    if(this.threadConnection.currentPacket
+ 1 == this.threadConnection.numberOfPackets)
    {
        len = (int)
(this.threadConnection.fileSize - (this.threadConnection.numberOfPackets -
1) * Config.BUFFER_SIZE);
        data = Arrays.copyOfRange(data, 0,
len);
    }
    if(len !=
this.receiveDatagram.getLength())
        continue;

    try
    {
        fileOutputStream.write(data);
    }
    catch (IOException e) {
        try
        {
            msg = "-1";
            sendData =
msg.getBytes(StandardCharsets.UTF_8);
            sendDatagram = new
DatagramPacket(sendData, sendData.length);
            this.threadSocket.send(sendDatagram);
            this.threadSocket.receive(this.receiveDatagram);
            errorFlag = false;
            break;
        }
        catch (IOException subexception) {
}
}

```

```

null)
this.threadConnection.file.delete();

1; // Bytes

= -1;

this.threadConnection.numberOfPackets = -1;
this.threadConnection.streamState =
"off";

System.out.println("SERWER-
MSG(THREAD:" + this.threadName + "): Wystąpił błąd podczas zapisu
pobieranego pliku: " + fileName);

break;

}

this.threadConnection.currentPacket++;
if(this.threadConnection.currentPacket
== this.threadConnection.numberOfPackets)
{
    this.threadConnection.file = null;
    this.threadConnection.fileSize = -
1; // Bytes

    this.threadConnection.currentPacket
= -1;

    this.threadConnection.numberOfPackets = -1;
    this.threadConnection.streamState =
"off";

    try
    {
        msg = "0";
        sendData =
msg.getBytes(StandardCharsets.UTF_8);

        sendDatagram = new
DatagramPacket(sendData, sendData.length);

        this.threadSocket.send(sendDatagram);

        this.threadSocket.receive(this.receiveDatagram);

        errorFlag = false;
        if(fileOutputStream != null)
            fileOutputStream.close();
        fileOutputStream = null;
        System.out.println("SERWER-
MSG(THREAD:" + this.threadName + "): Pobieranie pliku o nazwie: " +
fileName + " zakończyło się pomyślnie.");

        break;

    }
    catch (IOException subexception) {

        break;

    }

}

}

catch (IOException ignored) {
    this.threadSocket.disconnect();

```

```

        }
        }
        this.threadSocket.close();
    }
}
System.gc();
}

public class Server
{
    public static void main(String[] args) {
        String startPath = Paths.get("").toAbsolutePath().toString();
        DatagramSocket socket = null;
        byte[] receiveData;
        DatagramPacket receiveDatagram;
        ArrayList<Connection> connections = new ArrayList<Connection>();

        try {
            socket = new DatagramSocket(Config.PORT);
        }
        catch (SocketException exception)
        {
            System.out.println("SERWER-ERROR: Nie udało utworzyć się
głównego gniazda UDP dla serwera.");
            System.out.println(exception.getMessage());
            System.exit(-1);
        }
        System.out.println("SERWER-MSG: Serwer został uruchomiony
poprawnie.");

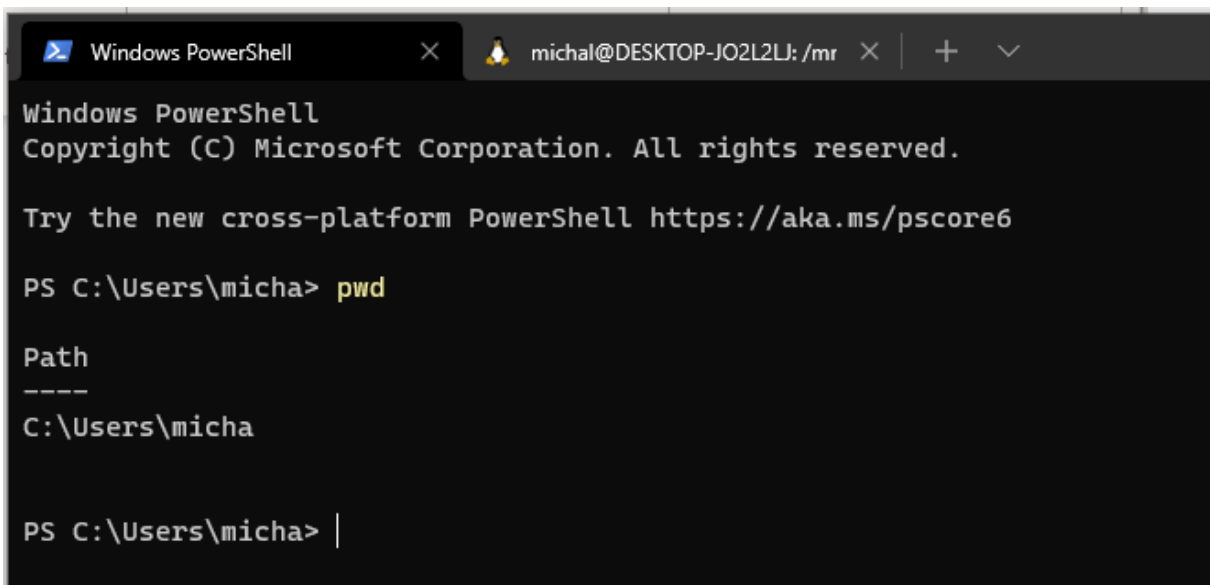
        while(true)
        {
            receiveData = new byte[Config.MAX_BUFFER_SIZE];
            receiveDatagram = new DatagramPacket(receiveData,
receiveData.length);
            try {
                socket.receive(receiveDatagram);
                new
ServerThread(receiveDatagram.getSocketAddress().toString(), connections,
receiveDatagram, startPath);
            }
            catch (IOException exception)
            {
                System.out.println("SERWER-ERROR: Nie udało odebrać się
wiadomości od klienta.");
                System.out.println(exception.getMessage());
            }
        }
    }
}

```

4. Wyjaśnienie sposobu rozwiązania podstawowych problemów.

```
import java.nio.file.Path;  
import java.nio.file.Paths;
```

Użycie powyższych klas umożliwia pozbycie się problemu związanego z konwencją stosowanego zapisu ścieżek plików.



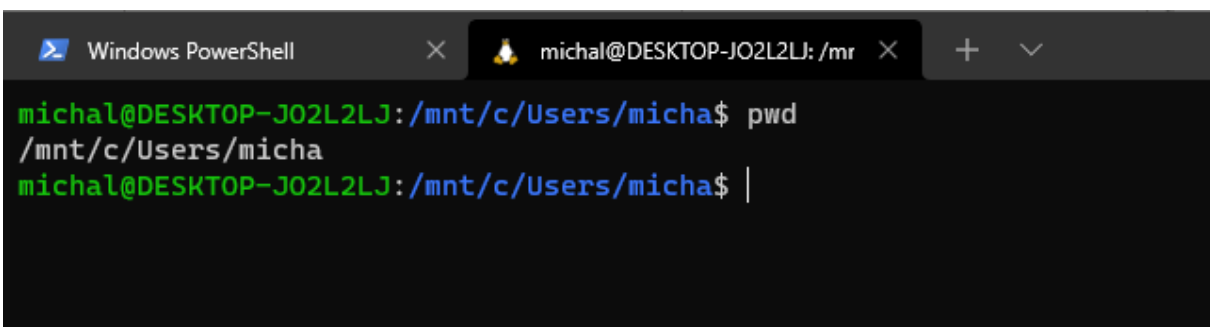
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\micha> pwd

Path
----
C:\Users\micha

PS C:\Users\micha> |
```



```
michal@DESKTOP-JO2L2LJ: /mnt/c/Users/micha$ pwd
/mnt/c/Users/micha
michal@DESKTOP-JO2L2LJ: /mnt/c/Users/micha$ |
```



```

if (socket == null) {
    try {
        socket = new DatagramSocket();
        socket.setSoTimeout(Config.TIMEOUT_MILLISECONDS);
    }
}

```

```

boolean errorFlag = true;
for(int i = 0; i < Config.TIMEOUT_TIMES; i++)
{
    try {
        socket.send(sendDatagram);
        socket.receive(receiveDatagram);
        errorFlag = false;
        break;
    }
    catch (IOException exception) {
    }
}
if(errorFlag)
{
    System.out.print("Nie udało nawiązać się połączenia z serwerem.");
    connectionState = false;
}
else

```

Dzięki przypisaniu gniazdu czasu oczekiwania, można próbować przesłać żądanie kilkakrotnie do czasu otrzymania odpowiedzi, bez obawy, że powodem błędu komunikacji jest zgubienie datagramu, a nie aktualny realny brak połączenia klienta z serwerem.

```

try
{
    address = new InetSocketAddress(InetAddress.getByName(currentIp), Config.PORT);
}
catch (UnknownHostException exception)
{
    System.out.print("Podano nieprawidłowy adres ip serwera.");
}

```

Poprawność adresu ip jest sprawdzana, dla nie prawidłowego adresu rzuca wyjątek, który następnie zostaje obsłużony.

```
if(!this.threadConnection.file.exists())
```

```
if(this.threadConnection.file.isDirectory())
```

```
if(!this.threadConnection.file.canRead())
```

```
try
{
    file.delete();
    file.createNewFile();
    if(!file.canWrite())
        throw new IOException();
    fileOutputStream = new FileOutputStream(file);
}
```

Uprawnienia do pliku, istnienie pliku, a także czy jest on katalogiem, sprawdzane są za pomocą powyższych metod.

```
try
{
    String tokens[] = msg.split( regex: " ");
    Long receiveCurrentPacket = Long.parseLong(tokens[0]);
    Long receiveNumberOfPackets = Long.parseLong(tokens[1]);
    Long receiveFileSize = Long.parseLong(tokens[2]);
    int len = Config.BUFFER_SIZE;

    if (receiveCurrentPacket + 1 == receiveNumberOfPackets)
    {
        len = (int) (receiveFileSize - (receiveNumberOfPackets - 1) * Config.BUFFER_SIZE);
    }

    if (currentPacket == -1) {
        fileData = new byte[len];
        fileInputStream.read(fileData, off: 0, len);
        currentPacket = 0;
    }

    if (currentPacket < receiveCurrentPacket) {
        fileData = new byte[len];
        fileInputStream.read(fileData, off: 0, len);
        currentPacket = receiveCurrentPacket;
    }

    sendDatagram = new DatagramPacket(fileData, fileData.length, receiveDatagram.getSocketAddress());
}
```

Odczyt pliku do wysłania odbywa się za pomocą klasy FileInputStream. Strona wysyłająca przy pierwszej próbie żądania wysyła rozmiary i ilość stron pliku, a następnie oczekuje na żądanie strony pobierającej plik. Strona pobierająca na początku pobiera pierwszą stronę pliku, a następnie każdą kolejną porcję danych pobiera, kiedy aktualny numer strony pliku strony wysyłającej jest większy od

aktualnej strony pliku strony wysyłającej. Rozmiarem ostatniej strony jest różnica rozmiaru pliku, a rozmiaru bufora pomnożonego przez liczbę wszystkich stron – 1.

```
int len = Config.BUFFER_SIZE;
byte[] data = this.receiveDatagram.getData();
if(this.threadConnection.currentPacket + 1 == this.threadConnection.numberOfPackets)
{
    len = (int) (this.threadConnection.fileSize - (this.threadConnection.numberOfPackets - 1) * Config.BUFFER_SIZE);
    data = Arrays.copyOfRange(data, 0, len);
}

if(len != this.receiveDatagram.getLength())
    continue;

try
{
    fileOutputStream.write(data);
}
```

Zapis pobieranego pliku odbywa się za pomocą klasy FileOutputStream. Strona pobierająca plik po wysłaniu żądania o kolejną porcję danych, sprawdza czy rozmiar datagramu odpowiada oczekiwanemu rozmiarowi strony pliku. Jeśli rozmiar jest prawidłowy, próbuje dopisać dane do pliku, jeśli nie wysła ponowną prośbę, o otrzymanie tej samej strony pliku.

```
while(true)
{
    receiveData = new byte[Config.MAX_BUFFER_SIZE];
    receiveDatagram = new DatagramPacket(receiveData, receiveData.length);
    try {
        socket.receive(receiveDatagram);
        new ServerThread(receiveDatagram.getSocketAddress().toString(), connections, receiveDatagram, startPath);
    }
    catch (IOException exception)
    {
        System.out.println("SERWER-ERROR: Nie udało odebrać się wiadomości od klienta.");
        System.out.println(exception.getMessage());
    }
}
```

Serwer po otrzymaniu datagramu od klienta, przekazuje informacje do nowego wątku, który powinien zająć się obsługą tego żądania.

Reprezentacją połączenia z klientem jest klasa Connection, której nazwą jest adres ip i portu klienta.

```
class Connection {
    public String connectionName;
    public String startPath;
    public String currentPath = "";

    public FileInputStream threadFileInputStream = null;
    public File file = null;
    public byte[] fileData = null;
    public long fileSize = -1; // Bytes
    public long currentPacket = -1;
    public long numberOfPackets = -1;
    public String streamState = "off"; // off, download or upload

    public Connection(String connectionName, String startPath) {
        this.connectionName = connectionName;
        this.startPath = startPath;
    }

    public void disconnect()
    {
        if(this.threadFileInputStream != null)
        {
            try {
                this.threadFileInputStream.close();
            }
            catch (IOException exception) {}
            this.threadFileInputStream = null;
        }
        if(this.file != null && this.streamState.equals("upload") && this.currentPacket + 1 < this.numberOfPackets)
            this.file.delete();
        this.file = null;
        System.out.println("SERVER-MSG(THREAD:" + this.connectionName + "): Zakończono połączenie z klientem.");
    }
}
```

Taką samą nazwę otrzymuje nowo tworzony wątek do obsługi klienta, dzięki czemu wątek ten może znaleźć odpowiadające mu połączenie klienta. (Każdy wątek tworzy tymczasowe gniazdo służące do odesłania informacji do klienta.

```
public void run()
{
    if(!connections.isEmpty())
    {
        for(int i = 0; i < connections.size(); i++)
        {
            if(this.threadName.equals(connections.get(i).connectionName))
            {
                this.threadConnection = connections.get(i);
                this.foundFlag = true;
                break;
            }
        }
    }
}
```

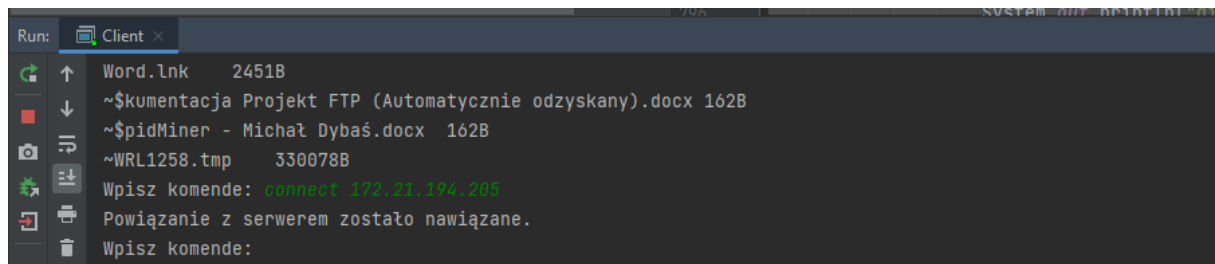
5. Działanie aplikacji

```
C:\Programy\Java\jdk\bin\java.exe "-javaagent:C:\Programy\JetBrains\IntelliJ IDEA Community Edition 2021.1\lib\idea_rt.jar=53672:C:\Programy\JetBrains\IntelliJ IDEA Community Edition 2021.1\bin" -Dfile.encoding=UTF-8 -classpath
Witaj w aplikacji, aby zobaczyć listę dostępnych komend wpisz help.
Wpisz komendę: help
Dostępne komendy:
help -- wyświetla listę komend
help 'nazwa_komendy' -- wyświetla informacje na temat danej komendy
dir -- wyświetla listę plików i katalogów w aktualnej ścieżce
pwd -- wyświetla aktualną ścieżkę
cd -- przechodzi lub wychodzi z katalogu
path 'nowa_ściezka' -- przechodzi do podanej ścieżki
connect 'adres_ip_serwera' -- nawiązuje połączenie z serwerem o podanym adresie
address -- wyświetla informacje na temat aktualnego adresu ip serwera
status -- wyświetla informacje na temat aktualnego połączenia z serwerem
sdir -- wyświetla listę plików możliwych do pobrania z serwera
download 'nazwa_pliku' -- pobiera plik z serwera do aktualnego katalogu roboczego
upload 'nazwa_pliku' -- wysyła plik z aktualnego katalogu roboczego na serwer
exit -- wychodzi z programu
Wpisz komendę: 'adres_ip_serwera' -- próbuje nawiązać połączenie z serwerem o podanym adresie ip
Wpisz komendę: cd .. -- polecenie to przechodzi do katalogu nadrzędnego jeśli to możliwe
cd ~ -- polecenie to przechodzi do startowego katalogu roboczego
cd 'nazwa_katalogu' -- polecenie to przechodzi do podkatalogu o podanej nazwie.
Wpisz komendę: Aktualna ścieżka katalogu roboczego: C:\Users\micha\Desktop\III Semestr\Latowozpraszane systemy baz danych\Projekt Java FTP\src
Wpisz komendę: C:\Users\
Aktualna ścieżka katalogu roboczego: C:\Users
Wpisz komendę: C:\Users\micha
Aktualna ścieżka katalogu roboczego: C:\Users\micha
Wpisz komendę: C:\Users\micha\Desktop
Aktualna ścieżka katalogu roboczego: C:\Users\micha\Desktop
Wpisz komendę: .vs katalog 0B
Android Studio.lnk 1000B
desktop.ini 282B
Discord.lnk 2231B
Dokumentacja Projekt FTP (Automatycznie odzyskany).docx 408348B
Excel.lnk 2413B
Harmonogram zajęć stacjonarne.pdf 256815B
```

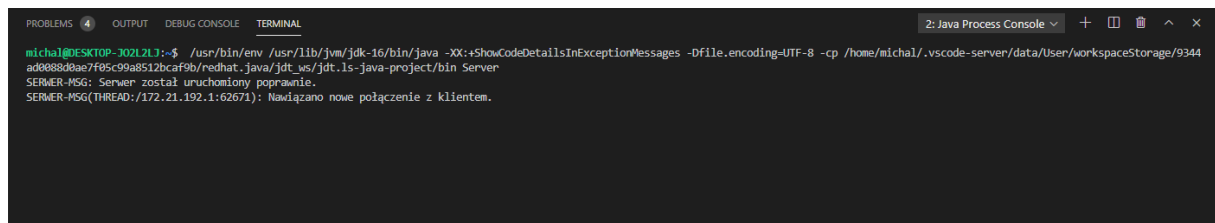
Rysunek 1 Aplikacja klienta uruchamiana w środowisku Windows

```
Windows PowerShell
michal@DESKTOP-JO2L2LJ: /mnt/c/Users/micha$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: bond0: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 02:42:de:f8:5f:96 brd ff:ff:ff:ff:ff:ff
3: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 42:29:ad:1e:bc:c5 brd ff:ff:ff:ff:ff:ff
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:af:76:c9 brd ff:ff:ff:ff:ff:ff
    inet 172.21.194.205/20 brd 172.21.207.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:feaf:76c9/64 scope link
        valid_lft forever preferred_lft forever
5: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
michal@DESKTOP-JO2L2LJ: /mnt/c/Users/micha$ |
```

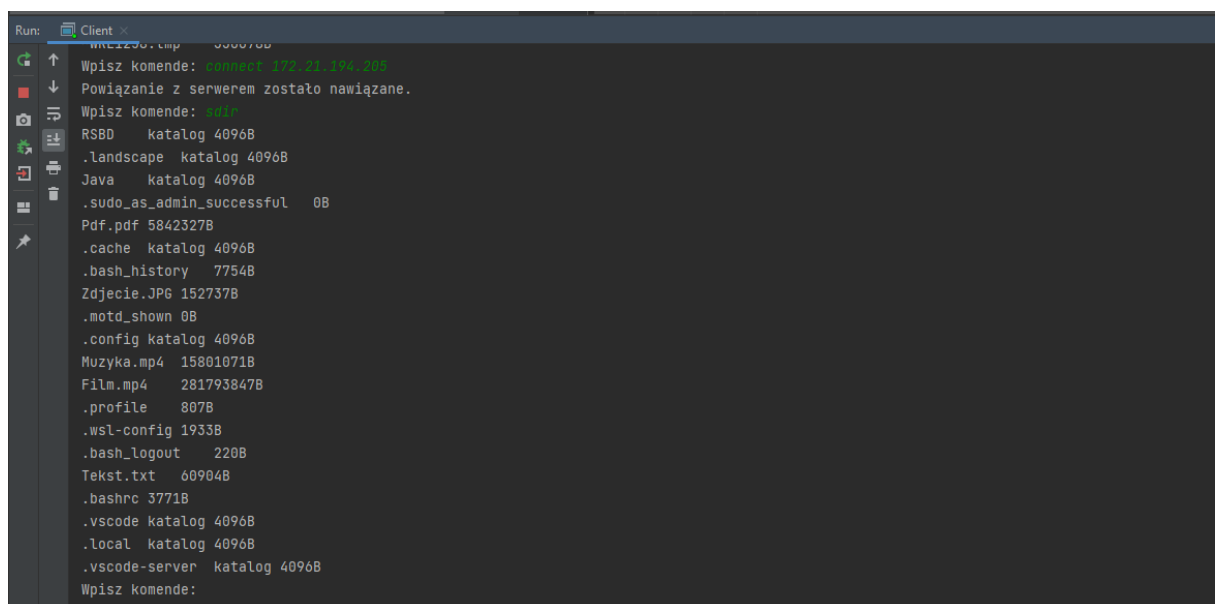
Rysunek 2 Adres ip systemu wsl Ubuntu (Windows Subsystem for Linux)



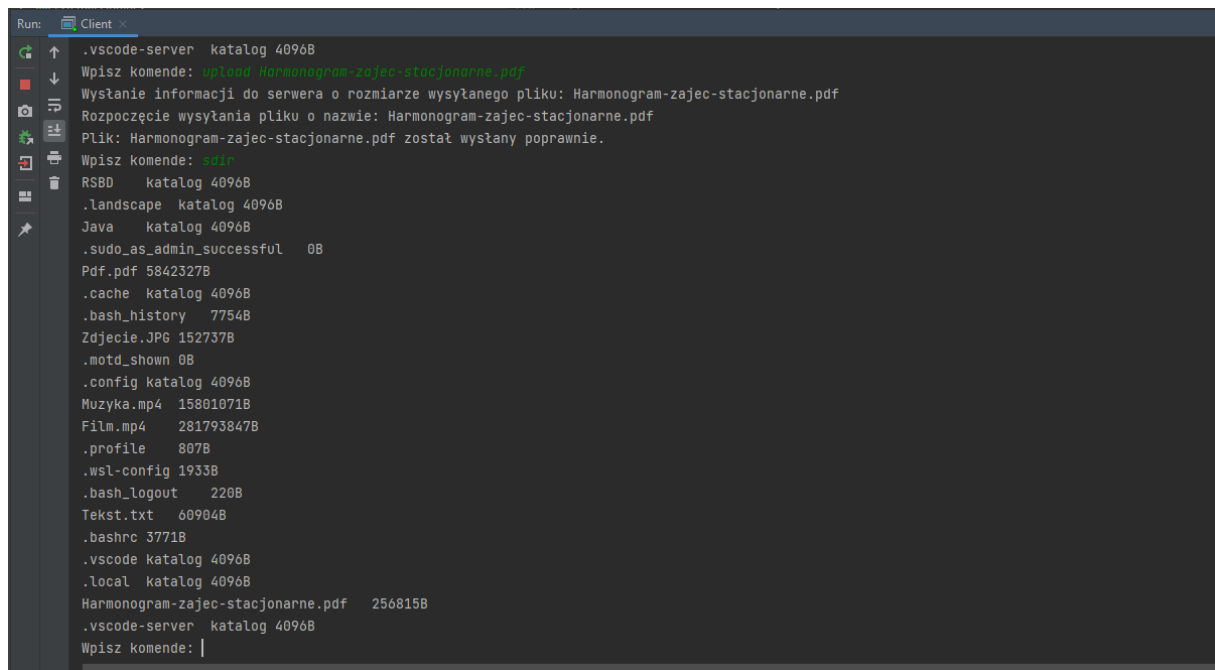
Rysunek 3 Połączenie się z serwerem uruchomionym na Ubuntu



Rysunek 4 Informacja o połączeniu pojawiła się również na konsoli serwera.

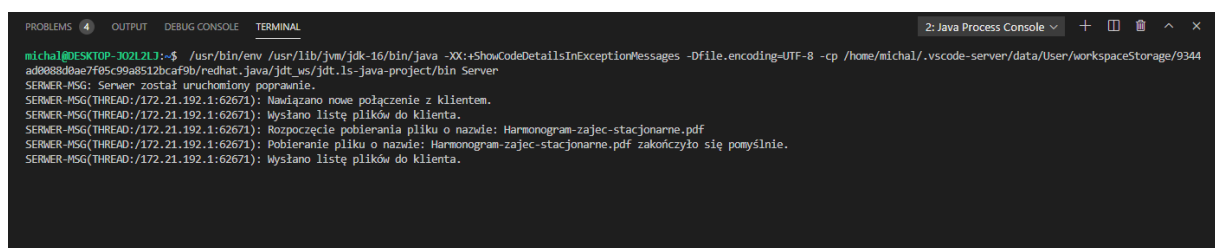


Rysunek 5 Wyświetlenie listy plików na serwerze



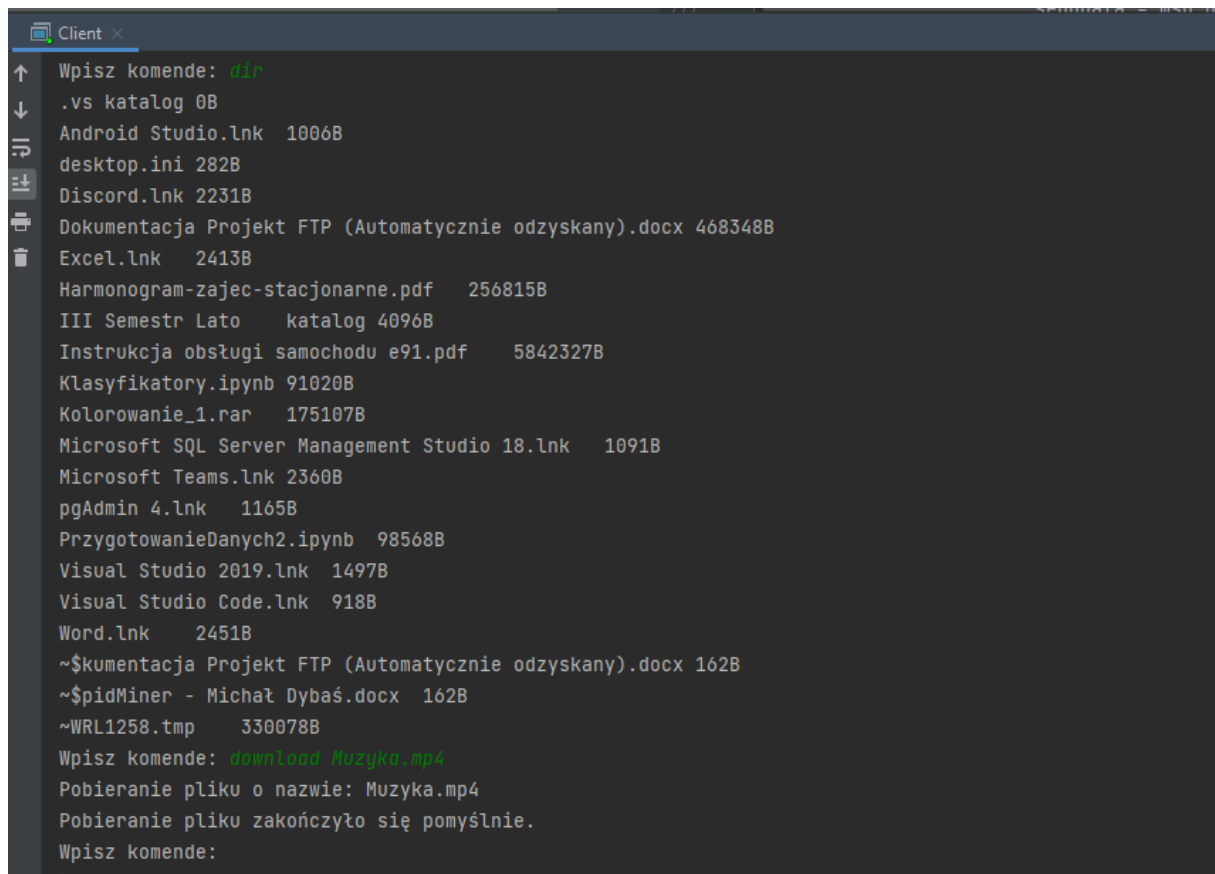
```
Run: Client
.vscode-server katalog 4096B
Wpisz komende: cat Harmonogram-zajec-stacjonarne.pdf
Wysłanie informacji do serwera o rozmiarze wysłanego pliku: Harmonogram-zajec-stacjonarne.pdf
Rozpoczęcie wysyłania pliku o nazwie: Harmonogram-zajec-stacjonarne.pdf
Plik: Harmonogram-zajec-stacjonarne.pdf został wysłany poprawnie.
Wpisz komende: cat
RSBD katalog 4096B
.landscape katalog 4096B
Java katalog 4096B
.sudo_as_admin_successful 0B
Pdf.pdf 5842327B
.cache katalog 4096B
.bash_history 7754B
Zdjecie.JPG 152737B
.motd_shown 0B
.config katalog 4096B
Muzyka.mp4 15801071B
Film.mp4 281793847B
.profile 807B
.wsl-config 1933B
.bash_logout 220B
Tekst.txt 60904B
.bashrc 3771B
.vscode katalog 4096B
.local katalog 4096B
Harmonogram-zajec-stacjonarne.pdf 256815B
.vscode-server katalog 4096B
Wpisz komende: |
```

Rysunek 6 Wysłanie pliku Harmonogram-zajec-stacjonarne.pdf na serwer



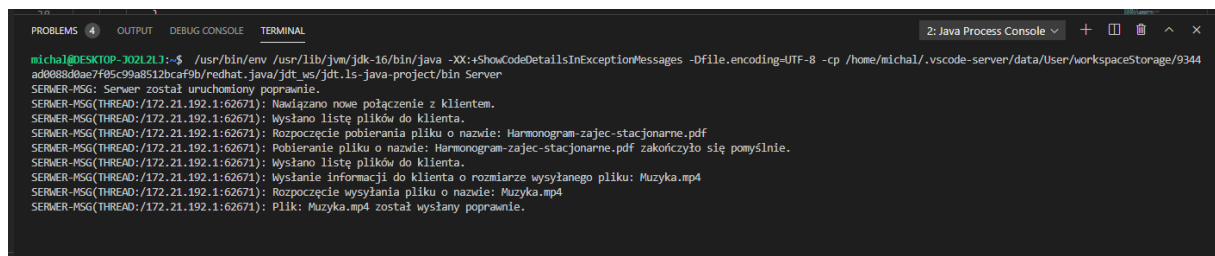
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Java Process Console
michal@DESKTOP-702L21J:~$ /usr/bin/env /usr/lib/jvm/jdk-16/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp /home/michal/.vscode-server/data/User/workspaceStorage/9344ad0888d8ae7f85c99a8512bc9f9b/redhat.java/jdt_vs/jdt.ls-java-project/bin Server
SERVER-MSG: Server został uruchomiony poprawnie.
SERVER-MSG(THREAD:/172.21.192.1:62671): Nawiązano nowe połączenie z klientem.
SERVER-MSG(THREAD:/172.21.192.1:62671): Wysłano listę plików do klienta.
SERVER-MSG(THREAD:/172.21.192.1:62671): Rozpoczęcie pobierania pliku o nazwie: Harmonogram-zajec-stacjonarne.pdf
SERVER-MSG(THREAD:/172.21.192.1:62671): Pobieranie pliku o nazwie: Harmonogram-zajec-stacjonarne.pdf zakończyło się pomyślnie.
SERVER-MSG(THREAD:/172.21.192.1:62671): Wysłano listę plików do klienta.
```

Rysunek 7 Informację, na temat powyższych operacji pojawiły się w logach serwera



```
Client x
↑ Wpisz komendę: dir
↓
.vs katalog 0B
Android Studio.lnk 1006B
desktop.ini 282B
Discord.lnk 2231B
Dokumentacja Projekt FTP (Automatycznie odzyskany).docx 468348B
Excel.lnk 2413B
Harmonogram-zajec-stacjonarne.pdf 256815B
III Semestr Lato katalog 4096B
Instrukcja obsługi samochodu e91.pdf 5842327B
Klasyfikatory.ipynb 91020B
Kolorowanie_1.rar 175107B
Microsoft SQL Server Management Studio 18.lnk 1091B
Microsoft Teams.lnk 2360B
pgAdmin 4.lnk 1165B
PrzygotowanieDanych2.ipynb 98568B
Visual Studio 2019.lnk 1497B
Visual Studio Code.lnk 918B
Word.lnk 2451B
~$kumentacja Projekt FTP (Automatycznie odzyskany).docx 162B
~$pidMiner - Michał Dybaś.docx 162B
~WRL1258.tmp 330078B
Wpisz komendę: download Muzyka.mp4
Pobieranie pliku o nazwie: Muzyka.mp4
Pobieranie pliku zakończyło się pomyślnie.
Wpisz komendę:
```

Rysunek 8 Pobranie pliku Muzyka.mp4

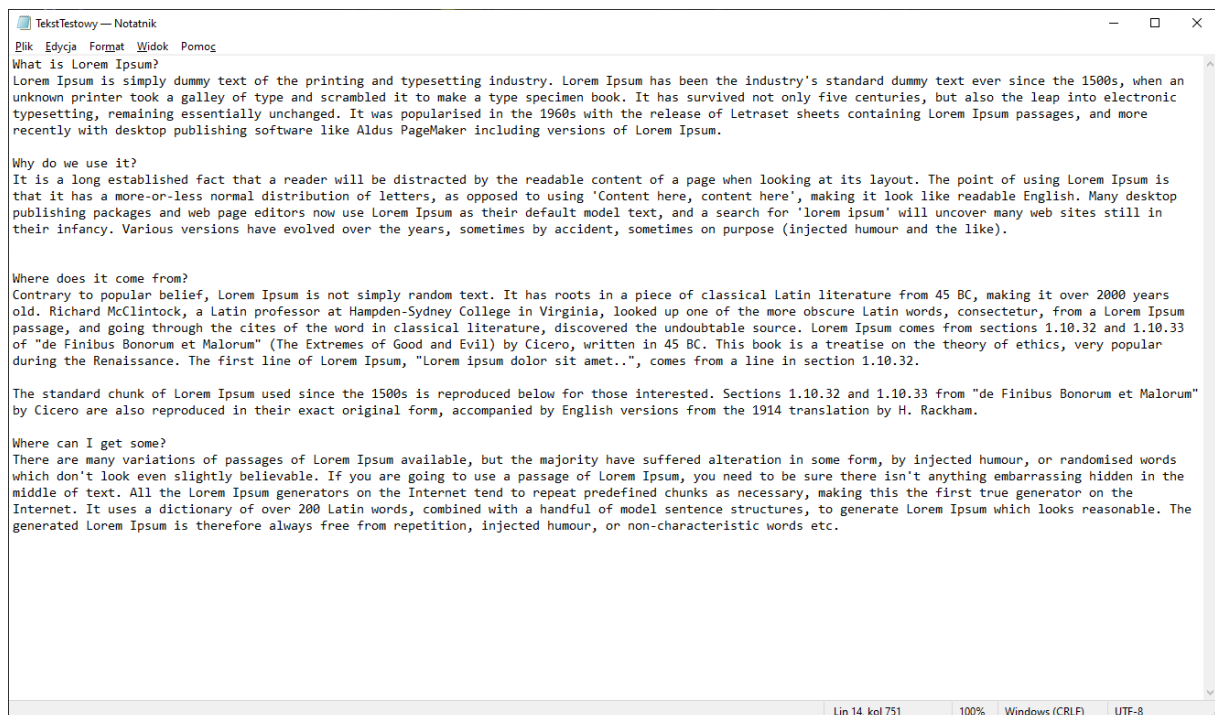


```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL 2: Java Process Console + - - x
michał@DESKTOP-302L2L1:~$ /usr/bin/env /usr/lib/jvm/jdk-16/bin/java -XX:+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp /home/michał/.vscode-server/data/User/workspaceStorage/9344
ad0080d0ae7f05c99a0512bc9f0b/redhat.java/jdt_ws/jdt.ls-java-project/bin Server
SERVER-MSG: Serwer został uruchomiony poprawnie.
SERVER-MSG(THREAD:/172.21.192.1:62671): Nawiązano nowe połączenie z klientem.
SERVER-MSG(THREAD:/172.21.192.1:62671): Wyślano listę plików do klienta.
SERVER-MSG(THREAD:/172.21.192.1:62671): Rozpoczęcie pobierania pliku o nazwie: Harmonogram-zajec-stacjonarne.pdf
SERVER-MSG(THREAD:/172.21.192.1:62671): Pobieranie pliku o nazwie: Harmonogram-zajec-stacjonarne.pdf zakończyło się pomyślnie.
SERVER-MSG(THREAD:/172.21.192.1:62671): Wyślano listę plików do klienta.
SERVER-MSG(THREAD:/172.21.192.1:62671): Wyślanie informacji do klienta o rozmiarze wysłanego pliku: Muzyka.mp4
SERVER-MSG(THREAD:/172.21.192.1:62671): Rozpoczęcie wysyłania pliku o nazwie: Muzyka.mp4
SERVER-MSG(THREAD:/172.21.192.1:62671): Plik: Muzyka.mp4 został wysłany poprawnie.
```

Rysunek 9 Informacja ta również pojawiła się w logach serwera.



Rysunek 10 Film z muzyką został przesłany prawidłowo.



Rysunek 11 Utworzenie przykładowego pliku tekstowego po stronie klienta


```
Windows PowerShell
michal@DESKTOP-J02L2LJ: ~
michal@DESKTOP-J02L2LJ:~$ dir
Film.mp4  Harmonogram-zajec-stacjonarne.pdf  Java  Muzyka.mp4  Pdf.pdf  RSBD  Tekst.txt  TekstTestowy.txt  Zdjecie.JPG
michal@DESKTOP-J02L2LJ:~$ cat TekstTestowy.txt
What is Lorem Ipsum?
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it?
It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

What is Lorem Ipsum?
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it?
It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).
```

Rysunek 14 Odczyt pliku przesłanego na serwer