Systemy rozproszone - raport z zadań z zajęć (Sockety)

Mateusz Pawłowicz

Informacje wstępne

Wykonane zadania bazowały na dostarczonym kodzie w językach Java i Python (w szczególności dotyczące połączenia bazującego na protokole UDP). Poniższy raport zawiera jedynie te fragmenty kodu, które zostały zmodyfikowane celem wykonania konkretnego zadania (w raporcie nie ma całego kodu, jest on dołączony w archiwum).

1 Zadanie 1

1.1 Cel zadania

Zadanie polegało na zrealizowaniu połączenia w dwie strony korzystając z protokołu UDP modyfikując kod klienta i serwera napisy w języku Java. Klient miał wysłać wiadomość do serwera, serwer miał mu odpowiedzieć.

1.2 Modyfikacja i wykonanie

Modyfikacja polegała na przechwyceniu przez serwer adresu oraz portu, z którego nadano wiadomość (w klasie *Datagram* znajdują się te pola). Następnie serwer korzystając z tych informacji wysyła na adres klienta odpowiedź. Wymagało to delikatnej modyfikacji klienta - po wysłaniu wiadomości oczekuje na gnieździe na odpowiedź od serwera.
Poniższe zrzuty ekranu pokazują najważniejsze zmiany w kodzie:

```
InetAddress clientAddress = receivePacket.getAddress();
int clientPort = receivePacket.getPort();
String msg = new String(receivePacket.getData());
System.out.println("received msg: " + msg);

// send reply to client (added)
byte[] sendBuffer = "PONG".getBytes();
DatagramPacket packetToClient = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);
socket.send(packetToClient);
System.out.println("Sent pong to client on: " + clientAddress.toString() + ":" + clientPort);
```

Rysunek 1: Zmodyfikowany kod serwera Java dla zadania 1

```java
// receive reply from server (added)
byte[] receiveBuffer = new byte[1024];
Arrays.fill(receiveBuffer, (byte)0);

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
socket.receive(receivePacket);

String msg = new String(receivePacket.getData());
InetAddress replyAddress = receivePacket.getAddress();
int replyPort = receivePacket.getPort();
System.out.println("Received: " + msg + " from: " + replyAddress.toString() + ":" + replyPort);
```

Rysunek 2: Zmodyfikowany kod klienta Java dla zadania 1

W zmodyfikowanym kodzie serwer przechwytuje adres i port klienta, a następnie wysyła tam wiadomość to treści *PONG*. Klient odbiera tą wiadomość i wypisuje ją na standardowe wyjście. Dodatkowo serwer wypisuje gdzie wysyła wiadomość, a klient skąd odebrał (adres oraz port).

1.3 Prezentacja działania

Poniższe zrzuty ekranu prezentują poprawne działanie programów:

```
JAVA UDP CLIENT
Received: PONG from: /127.0.0.1:9008
```

```
JAVA UDP SERVER
received msg: Ping Java Udp
Sent pong to client on: /127.0.0.1:65186
```

Rysunek 3: Wyniki działania klienta oraz serwera Java dla zadania 1

2 Zadanie 2

2.1 Cel zadania

Zadanie polegało na zrealizowaniu połączenia w dwie strony korzystając z protokołu UDP modyfikując kod klienta napisany w jęzku Python i serwera napisy w języku Java. Klient miał wysłać do serwera wiadomość zawierającą polskie znaki - *żółta gęś*, serwer miał mu odesłać tę samą wiadomość.

2.2 Modyfikacja i wykonanie

Modyfikacja względem poprzedniego zadania polegała na tym, że zarówno w klient jak i serwer kodował wysyłaną wiadomość oraz dekodował odbieraną przy pomocy kodowania UTF-8 (zawiera ono polskie znaki). Pozwoliło to na poprawne działanie.
Oto zrzuty ekranu pokazują najistotniejsze zmiany w kodzie:

```java
InetAddress clientAddress = receivePacket.getAddress();
int clientPort = receivePacket.getPort();
String msg = new String(receivePacket.getData(), StandardCharsets.UTF_8);
System.out.println("received msg: " + msg);

// send reply to client (added)
byte[] sendBuffer = ("PONG - " + msg).getBytes(StandardCharsets.UTF_8);
DatagramPacket packetToClient = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);
socket.send(packetToClient);
System.out.println("Sent pong to client on: " + clientAddress.toString() + ":" + clientPort);
```

Rysunek 4: Zmodyfikowany kod serwera Java dla zadania 2

```python
import socket

serverIP = "127.0.0.1"
serverPort = 9008
msg = "żółta gęś"

print("Python UDP client!")
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.sendto(bytes(msg, 'utf-8'), (serverIP, serverPort))

buff = []
buff, address = client.recvfrom(2048)
print("received: " + str(buff, "utf-8") + "from: " + str(address[0]) + ":" + str(address[1]))
```

Rysunek 5: Zmodyfikowany kod klienta Python dla zadania 2

2.3 Prezentacja działania

Poniższe zrzuty ekranu przedstawiają poprawne działanie programów:

```
Python UDP client!
received: PONG - żółta gęśfrom: 127.0.0.1:9008
```

```
JAVA UDP SERVER
received msg: żółta gęś
Sent pong to client on: /127.0.0.1:52297
```

Rysunek 6: Wyniki działania klienta Python oraz serwera Java dla zadania 2

3 Zadanie 3

3.1 Cel zadania

Zadanie polegało na zrealizowaniu połączenia w dwie strony korzystając z protokołu UDP modyfikując kod klienta napisany w języku Python i serwera napisy w języku Java. Klient wysłał liczbę typu *Integer* symulując posiadanie innej architektury (little-endian) niż serwer (big-endian). Serwer miał poprawnie odebrać liczbę, zinkrementować ją, a następnie wysłać ją do klienta (który też miał poprawnie ją odebrać).

3.2 Modyfikacja i wykonanie

Modyfikacja tego zadania polegała na wprowadzeniu dodatkowo interpretacji kolejności bajtów odebranych z gniazda. Jako, że zgodnie z treścią, klient miał wysłać liczbę (4-bajtową) w kolejności little-endian , to serwer musiał odebrać

ją w też uwzględniając tą samą kolejność (oraz rozmiar). Następnie serwer inkrementował liczbę i odsyłał ją z powrotem do klienta, ponownie zamieniając na little-endian, klient również odbierał w kolejności little-endian (symulował posiadanie tej architektury). Alternatywnie, serwer mógłby wysłać bajty w kolejności big-endian, o ile klient też by je odbierał w tej kolejności.
Oto zrzuty ekranu pokazują najistotniejsze zmiany w kodzie:

```java
InetAddress clientAddress = receivePacket.getAddress();
int clientPort = receivePacket.getPort();
int receivedNumber = ByteBuffer.wrap(receivePacket.getData()).order(ByteOrder.LITTLE_ENDIAN).getInt();
System.out.println("received number: " + receivedNumber);

// send reply to client (added)
receivedNumber++;
byte[] sendBuffer = ByteBuffer.allocate(4).order(ByteOrder.LITTLE_ENDIAN).putInt(receivedNumber).array();
DatagramPacket packetToClient = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);
socket.send(packetToClient);
System.out.println("Sent incremented number to client on: " + clientAddress.toString() + ":" + clientPort);
```

Rysunek 7: Zmodyfikowany kod serwera Java dla zadania 3

```python
import socket

serverIP = "127.0.0.1"
serverPort = 9008
msg = (69).to_bytes(4, byteorder="little")

print("Python UDP client!")
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.sendto(msg, (serverIP, serverPort))

buff = []
buff, address = client.recvfrom(2048)
received_number = int.from_bytes(buff, byteorder='little')
print("received: " + str(received_number) + " from: " + str(address[0]) + ":" + str(address[1]))
```

Rysunek 8: Zmodyfikowany kod klienta Python dla zadania 3

3.3 Prezentacja działania

Na poniższych zrzutach ekranu widać poprawne działanie programów:

```
Python UDP client!
received: 70 from: 127.0.0.1:9008
```

```
JAVA UDP SERVER
received number: 69
Sent incremented number to client on: /127.0.0.1:61497
```

Rysunek 9: Wyniki działania klienta Python oraz serwera Java dla zadania 3