POLITECHNIKA POZNAŃSKA
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Alicja Mruk
140752

Kamil Osak
140756

Dokumentacja projektu sieciowego
WARIANT 19 – MODEL KOMUNIKACJI N↔1 PROTOKÓŁ TEKSTOWY

Grupa L6

12 listopada 2019

1. Treść zadania

- Protokół warstwy transportowej: UDP
- Pola nagłówka protokołu tekstowego zdefiniowane jako `klucz#wartość@`
- Nazwy pól o określonej długości: 4 znaki
- Podstawowe pola nagłówka oraz odpowiadające im klucze:
  - pole operacji – „oper”
  - pole statusu – „stat”
  - pole identyfikatora sesji – „iden”
  - dodatkowe pola zdefiniowane przez programistę – zgodnie z wymaganiami
- Funkcje oprogramowania:
  - uzgodnienie identyfikatora sesji
  - wykonywanie operacji na trzech argumentach:
    - „mnozenie” – mnożenie
    - „dodawanie” – dodawanie
    - 2 inne, dowolnie wybrane operacje matematyczne
  - wykonywanie operacji sumowania wielu liczb:
    - klient przesyła kolejne wartości liczbowe wraz z informacją, czy jest to ostatni komunikat
    - serwer każdorazowo potwierdza otrzymanie danych osobnym komunikatem i zwraca wynik
- Wymagania dodatkowe:
  - identyfikator sesji oraz znacznik czasu powinny być przesyłane w każdym komunikacie
  - serwer powinien obsługiwać i rozróżniać wielu klientów jednocześnie.

2. Opis protokołu

- pola zostały zdefiniowane jako `klucz#wartość@`
- oprócz trzech podstawowych pól, zdefiniowano nowe pole - „`time`”
- jako wymaganie dodatkowe dodano identyfikator sesji dla każdego klienta, przysyłany wraz ze znacznikiem czasu w każdym komunikacie
- komunikat składa się z 4 pól
  - `stat` – pole statusu – zawiera identyfikator odpowiedzi serwera, dla klienta jest to zawsze wartość `null`
  - `oper` - pole operacji, która powinna zostać wykonana po stronie serwera
    - możliwe operacje
    - dodawanie
      - wybór 2 opcji sumowania w menu wyboru
        - sumowanie 3 liczb
        - sumowanie n liczb
    - odejmowanie
    - mnozenie
    - dzielenie
    - zakończenie połączenia

- iden – pole wartości liczbowej - zawiera identyfikator danej sesji i liczby, które zostaną użyte w danej operacji
- time – pole znacznika czasu - ilość milisekund, które upłynęły od 01.01.1970r.

- długość komunikatu jest zmienna (zależy od transmitowanych danych)
  - opcjonalne pola od których zależy długość komunikatu :
    - result - pole wyniku zwracane przez serwer
    - num$ - pola z liczbami przekazywanymi przez serwer ($ - to dowolna liczba naturalna)

Przykładowe komunikaty wysłane przez Klienta:

1) Komunikat wysyłany przez klienta w momencie pierwszego połączenia się z serwerem.

Wysyłane jest zapytanie do serwera o przydzielenie ID aktualnej sesji

`oper#getid@stat#null@iden#null@time#1572976627558@`

- `oper#getid@-`zapytanie skierowane do serwera o przydzielenie numeru sesji
- `stat#null@-`pole statusu dla klienta zawsze wynosi `null`
- `iden#null@-`aktualny numer sesji jest nieznany przez klienta
- `time#15729766278-`ilość ms, które upłynęły od 01.01.1970r.

2) Wysłanie żądania operacji

`oper#dodawanie@stat#null@iden#0@num1#2@num2#3@num3#5@time#1576697@`

- `oper#dodawanie@-`wybrano operację dodawania
- `stat#null@`
- `iden#0@num1#2@num2#3@num3#5@-`identyfikator sesji wynosi 0

podano trzy liczby do wykonania operacji dodawania: 2, 3, 5

- `time#1576697@`

3. Zakończenie sesji

`oper#close@stat#null@iden#0@time#1572977414834@`

- `oper#close@-`zakończenie sesji
- `stat#null@`
- `iden#0@-`identyfikator sesji klienta wynosi 0
- `time#1572977414834@`

Przykładowe komunikaty wysyłane przez Serwer:

1. Komunikat wysłany przez serwer w momencie, gdy w otrzymanym wcześniej komunikacie zawarta została prośba i identyfikator klienta

`oper#setid@stat#ok@iden#0@time#1572976627558@`

- `oper#setid@` - operacja ustawienia identyfikatora dla klienta
- `stat#ok@` - serwer poprawnie odebrał komunikat od Klienta
- `iden#0@` - przydzielony numer sesji dla danego klienta
- `time#1572976627558@`

2. Wysłanie rezultatu działania

`oper#dodawanie@stat#ok@iden#0@result#10@time#1576697@`

- `oper#dodawanie@` - operacja dodawania (możliwe wybory: dodawanie, odejmowanie, dzielenie, mnozenie)
- `stat#ok@`
- `iden#0@` - numer sesji klienta przesyłającego datagram wraz z operacją i danymi
- `time#1576697@`

3. Zwolnienie identyfikatora dla kolejnych Klientów

`oper#releaseid@stat#ok@iden#null@time#1572977414834@`

- `oper#releaseid@` - operacja zwalniania identyfikatora sesji w momencie zakończenia połączenia z serwerem przez Klienta
- `stat#ok@`
- `iden#null@` - pole identyfikatora sesji wskazujące, że identyfikator dla danego klienta został ustawiony na `null` (po wcześniejszym zwolnieniu identyfikatora)
- `time#1572977414834@`

3. Aplikacja użytkownika (klienta) oraz aplikacja serwera

1. Klient

   1) Tworzenie komunikatu

   Klasą odpowiedzialną za tworzenie komunikatu jest klasa `Operacja` i funkcja w niej zawarta: `getKomunikat()`

   Funkcja jest wywoływana po wyświetleniu na ekranie menu wyboru. W zależności od wyboru klienta wykonywana jest konkatenacja dwóch łańcuchów znaków, pola `OPER` wraz z odpowiednim działaniem matematycznym.

   Funkcja `getLiczby()` odpowiada za pobranie od klienta liczb wykorzystywanych w danym działaniu. W przypadku wykonywania dodawania, wywoływana jest funkcja `menuIloscLiczb()` - użytkownik ma do wyboru dodanie 3 lub n liczb.

```
public String getKomunikat(){
IDEN += id + "@";
TIME += Czas.getGodzina() + "@";
    if (wybor == 0) {
        OPER += "close@";
    } else if (wybor == 1)
        OPER += "dodawanie@";
        sumFlag = true;
        menuIloscLiczb();
    } else if (wybor == 2) {
        OPER += "odejmowanie@";
        getLiczby();
    } else if (wybor == 3) {
        OPER += "mnozenie@";
        getLiczby();
    } else if (wybor == 4) {
```

```
        OPER += "dzielenie@";
        dzielenie = true;
        getLiczby();
    }
    if (OPER.equals("oper#close@")) {
        komunikat = OPER + STAT + IDEN + TIME;
    }
    else {
         if (sumFlag && iloscLiczb != 3){
         komunikat = OPER + STAT + IDEN;
           for (int i = 0; i < sum_n.size(); i++) {
         komunikat += "num" + (i + 1) + "#" + sum_n.get(i) + "@";
           }
        } else {
          komunikat = OPER + STAT + IDEN + NUMS[0] + NUMS[1] NUMS[2];
        }
        komunikat += TIME;
    }
        setDefaultTextOfStatement();

    return komunikat;
}

private static void menuIloscLiczb() {
         System.out.println("Wpisz 'a' jeśli chcesz wpisać trzy
liczby");
    System.out.println("Wpisz 'n' jeśli chcesz wpisać n liczb");
         char opcja = userEntry.next().charAt(0);
    if (opcja == 'a') {
         iloscLiczb = 3;
               }
         else if (opcja == 'n') {
               setIloscLiczb();

    }
  getLiczby();
}
```

2) Wysłanie i odebranie datagramu

Tworzona jest instancja klasy `Operacja`. Jej konstruktor zawiera ID aktualnej sesji. Nastepnie wyświetlane jest menu wyboru. Zmienna `choose` przechowuje aktualny wybór użytkownika.

(ukazano fragment funkcji)

```
Operacja operacja = new Operacja(ID_USER);
operacja.pokazMenu();
choose = operacja.getWybor();
messageToSend = operacja.getKomunikat();
```

```
sendToPacket = new DatagramPacket(messageToSend.getBytes(),
messageToSend.length(), IPAdress, PORT);
datagramSocket.send(sendToPacket);
receivedPacket = new DatagramPacket(buffer, buffer.length);
datagramSocket.receive(receivedPacket);
serverResponse = new String(receivedPacket.getData(),
                            0, receivedPacket.getLength());
```

2. Serwer

1) Otworzenie portu

Port otwierany jest poprzez wywołanie konstruktora obiektu klasy DatagramSocket, a parametrem jest liczba całkowita typu `int`, czyli podanie numeru portu służącego do późniejszej komunikacji

(ukazano fragment funkcji)

```
System.out.println(“Otwieranie portu\n”);
try {
    datagramSocket = new DatagramSocket(PORT);
} catch (SocketException sockEx) {
    System.out.println("Błąd podczas otwierania portu");
   System.exit(1);
}
handleClient();
```

2) Interpretacja otrzymanego komunikatu

Zawartość otrzymanego datagramu serwer interpretuje przy użyciu wyrażeń regularnych, ponieważ wszelkie wartości klucza oper są unikalne i nie mogą wykonywać innych części kodu.

```
private void calculateResultAndGetOPERACJAString() {
if (Pattern.compile("dodawanie").matcher(Operacja.KOMUNIKAT).find()) {
    OPER += "dodawanie@";

    Pattern p = Pattern.compile("(num\\d+#-?\\d+@)+");
   Matcher m = p.matcher(Operacja.KOMUNIKAT);

    if(m.find()) {
          p = Pattern.compile("-?\\d+");
          m = p.matcher(m.group());

int count = 0;
    while(m.find()) {
          count++;
          if (count % 2 == 0) {
```

```
                int temp = Integer.parseInt(m.group());
                sum_n.add((long) temp);
            }
        }
      }
      RESU_V = 0;
        for (Long aLong : sum_n) {
            RESU_V += aLong;
        }
        RESU += RESU_V + "@";
    }
    else {
        int counter = 0;
        int counter_number = 0;

        /*regex wykrywajacy 3 liczby w otrzymanym komunikacie od klienta*/
        Pattern p = Pattern.compile("num1#-?\\d+@num2#-?\\d+@num3#-?\\d+@");
        Matcher m = p.matcher(Operacja.KOMUNIKAT);

        if (m.find()) {
            String temp = m.group();
            p = Pattern.compile("-?\\d+");
            m = p.matcher(temp);

            while (m.find()) {
                counter_number++;
                if (counter_number % 2 == 0) {
                    NUMS_V[counter] = Integer.parseInt(m.group());
                    counter++;
                }
            }
        }
        if (Pattern.compile("mnozenie").matcher(Operacja.KOMUNIKAT).find()) {
            RESU_V = NUMS_V[0] * NUMS_V[1] * NUMS_V[2];
            OPER += "mnozenie@";
        } else if
    (Pattern.compile("dzielenie").matcher(Operacja.KOMUNIKAT).find()) {
            RESU_V = NUMS_V[0] / NUMS_V[1] / NUMS_V[2];
            OPER += "dzielenie@";
        } else if
    (Pattern.compile("odejmowanie").matcher(Operacja.KOMUNIKAT).find()) {
            RESU_V = NUMS_V[0] - NUMS_V[1] - NUMS_V[2];
            OPER += "odejmowanie@";
        } else if
    (Pattern.compile("getid").matcher(Operacja.KOMUNIKAT).find()) {
            OPER += "setid@";
        } else if
    (Pattern.compile("close").matcher(Operacja.KOMUNIKAT).find()) {
            OPER += "releaseid@";
```

```
    } else if
(Pattern.compile("error").matcher(Operacja.KOMUNIKAT).find()) {
          OPER += "agree@";
    }
}
/*przekazanie wyniku do komunikatu*/
RESU += RESU_V + "@";
/*przekazanie wyniku do komunikatu*/
}
```

3) Tworzenie datagramu

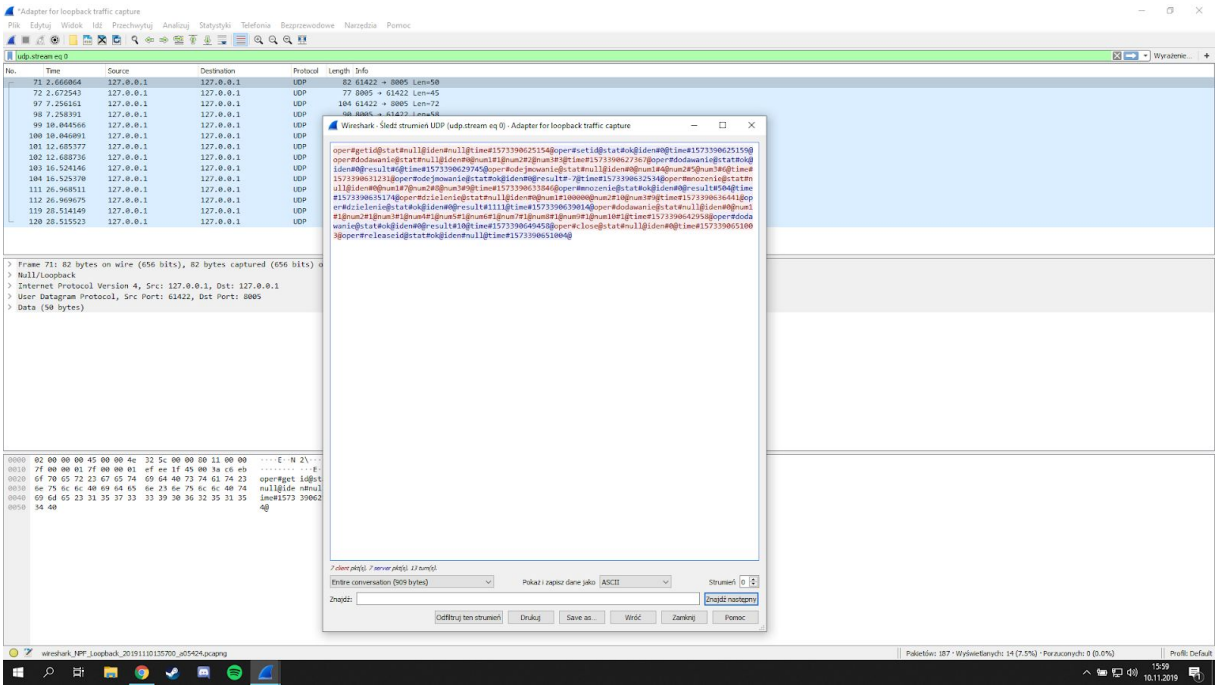
Datagramy tworzone są głównie na podstawie dwóch operacji: getid oraz close, ponieważ pozostałe operacje składają się z takich samych kluczy. Natomiast wyżej 2 wymienione zbudowane są z unikalnych kompozycji kluczy.

```
String createMessage() {
    STAT += "ok@";
    TIME+=Czas.getGodzina() + "@";

    calculateResultAndGetOPERACJAString();
    if(Pattern.compile("getid").matcher(Operacja.KOMUNIKAT).find()) {
          IDEN += UDPSerwer.getIdForUser() + "@";
          message = OPER + STAT + IDEN + TIME;

}
else if(Pattern.compile("close").matcher(Operacja.KOMUNIKAT).find()) {
    IDEN += "null@";
    message = OPER + STAT + IDEN + TIME;
}
else {
    IDEN += id_klient + "@";
    message = OPER + STAT + IDEN + RESU + TIME;
}
setDefaultTextOfStatement();
return message;
}
```

4. PRZYKŁADOWA TRANSMISJA W LOGACH PROGRAMU WIRESHARK



Rysunek 1 - logi przykładowej transmisji zarejestrowanej w programie Wireshark

1. Klient łączy się z serwerem i automatyczne wysyła prośbę o przyznanie numeru sesji (ID).

oper#getid@stat#null@iden#null@time#1573390625154@

2. Serwer odbiera datagram i odpowiada klientowi przyznając dany numer sesji.

oper#setid@stat#ok@iden#0@time#1573390625159@

3. Klient wysyła prośbę o wynik operacji dodawanie na 3 liczbach - 1, 2 i 3.

oper#dodawanie@stat#null@iden#0@num1#1@num2#2@num3#3@time#1573390627367@

4. Serwer otrzymuje datagram, rozpoznaje operacje, wykonuje stosowne działania i zwraca wynik w odpowiedzi - wynik: 6.

oper#dodawanie@stat#ok@iden#0@result#6@time#1573390629745@

5. Klient wysyła prośbę o wynik operacji odejmowanie na 3 liczbach: 4, 5 i 6.

oper#odejmowanie@stat#null@iden#0@num1#4@num2#5@num3#6@time#1573390631231@

6. Serwer otrzymuje datagram, rozpoznaje operacje odejmowania i wykonuje obliczenia. Po wszystkim, wynik zostaje zwrócony w datagramie wysłanym do klienta. Wynik: -7.

oper#odejmowanie@stat#ok@iden#0@result#-7@time#1573390632534@

7. Klient wysyła prośbę o wynik operacji mnożenie na 3 liczbach: 7, 8, 9.

oper#mnozenie@stat#null@iden#0@num1#7@num2#8@num3#9@time#1573390633846@

8. Serwer otrzymuje datagram, rozpoznaje i wykonuje operacje mnożenia. Wynik 504 zostaje zwrócony w datagramie.

oper#mnozenie@stat#ok@iden#0@result#504@time#1573390635174@

9. Klient wysyła prośbę o wynik dzielenia na 3 liczbach: 100000, 10 i 9. W przypadku wybrania operacji dzielenia niemożliwe jest wysłanie zera jako wartość liczby drugiej lub trzeciej. Jeśli zero zostanie wpisane na te miejsca, program poprosi o ponownie wpisanie liczby różnej od 0.

oper#dzielenie@stat#null@iden#0@num1#100000@num2#10@num3#9@time#1573390636441@

10. Serwer otrzymuje datagram, rozpoznaje działanie dzielenia. Wykonuje go i zwraca wynik w datagramie wysyłanym do klienta. Wynik to liczba całkowita: 1111.

oper#dzielenie@stat#ok@iden#0@result#1111@time#1573390639014@

11. Klient wysyła prośbę o wykonanie operacji dodawania na "n liczbach", w przypadku powyższej transmisji następuje wysłanie dziesięciu jedynek.

oper#dodawanie@stat#null@iden#0@num1#1@num2#1@num3#1@num4#1@num5#1@num6#1@num7#1@num8#1@num9#1@num10#1@time#1573390642958@

12. Serwer otrzymuje datagram i rozpoznaje operację dodawania "n liczb". Korzystając z dynamicznego kontenera, czyta liczby z bloku danych datagramu. Wynik 10 zostaje zwrócony w datagramie, który zostaje wysłany do klienta.

oper#dodawanie@stat#ok@iden#0@result#10@time#1573390649458@

13. Klient kończy sesję, wysyła do serwera datagram z operacją "close".

oper#close@stat#null@iden#0@time#1573390641003@

14. Serwer otrzymuje informację o zakończeniu sesji klienta. Zwalnia identyfikator sesji oraz wysyła klientowi ostatni datagram z informacją o zwolnieniu numeru sesji w danym czasie.

oper#releaseid@stat#ok@iden#null@time#1573390651004@

5. Odpowiedzi na pytania pozostawione w sekcji "Zadania szczegółowe"

a) Wyznacz numer wariantu zadania podlegającego implementacji (1). Zadanie należy wykonać w zespole 2-osobowym. *nr zadania* = ((*nr albumu* 1 + *nr albumu* 2) *mod* 30) + 1

nr zadania = ((140752 + 140756) mod 30) + 1 = (281508 mod 30) + 1 = 18 + 1 = 19

b) Przygotuj implementacje protokołu komunikacyjnego, aplikacji klienckiej oraz aplikacji serwerowej w wybranym języku wysokiego poziomu – C++, C#, Java lub Python

   Implementacja wariantu projektu sieciowego w języku Java znajduje się w załączniku w wiadomości email oraz na dysku sieciowym Google pod linkiem (dostępny tylko w formie elektronicznej).

c) Przetestuj połączenie pomiędzy programami, rejestrując pakiety danych. Przeanalizuj przechwycony ruch sieciowy. Określ, czy dane przesyłane są w postaci binarnej, czy tekstowej? Wskaż wady oraz zalety obu form komunikacji

   Przykładowe połączenie zostało przedstawione w punkcie 4 sprawozdania projektu sieciowego i rysunku nr 1.

   Dane w wariancie 19 projektu sieciowego przesyłane są w postaci tekstowej.

   Protokół binarny:

   - Przeznaczony do odczytu przez maszynę, a nie przez człowieka.
   - Wymaga przetworzenia lub zapoznania się z kodem opisującym znaczenia poszczególnych operacji, w celu jej interpretacji.
   - Jego największymi zaletami są
     - zwięzłość
     - szybkość transmisji i interpretacji
     - małe wymagania sprzętowe
     - proste i mało wymagające operacje

   Protokół tekstowy:

   - Jego cechą jest komunikat łatwy w interpretacji dla człowieka.
   - Nie wymaga przetworzenia, aby użytkownik był w stanie zrozumieć komunikat.
   - Analiza komunikatu i kontrola zawartości protokołu jest bezproblemowa, z racji na
   - strukturę pól komunikatu, definiowanych zwykle jako klucz-wartość.
   - Przykładem protokołu tekstowego jest HTTP.
   - Pochłania dużą ilość zasobów oraz jest skomplikowany z algorytmicznego punktu widzenia.

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

   W wariancie 19 brak domyślnej wielkości protokołu, lecz dla określenia teoretycznej wielkości można posłużyć się najprostszymi komunikatami z obowiązkowymi kluczami. Takimi komunikatami są:

   - oper#getid@stat#ok@iden#0@ - 26 znaków
   - oper#releaseid@stat#null@iden#0@ - 32 znaki

W wariancie 19 takie wyliczanie nie ma najmniejszego sensu, ponieważ wszystkie komunikaty są zmiennego rozmiaru, czasami dodawane są 4 klucze z wartościami, a czasami może być takich kluczy nawet 20, ponieważ w projekcie została zaimplementowana operacja dodawania “n liczb” - komunikat może przyjąć postać:

oper#dodawanie@stat#null@iden#10@num1#1@num2#2@num3#3@num4#4@num5#5@num6#4@num7#3@num8#2@num9#1@num10#0@time#123456789@

Podsumowując, rozmiar jest w pełni zależy od wybranej operacji i wprowadzonych danych. Maksymalny możliwy rozmiar komunikatu możliwego do odczytania zależny jest od rozmiaru bufora zastosowanego w kodzie.

Rozbudowa protokołu może być całkiem łatwa dla programisty, ponieważ wszelkie interpretacje kluczy i ich wartości odbywa się za pomocą wyrażeń regularnych jako odpowiednich warunków, wystarczy dodać nowy warunek i zaimplementować jego budowę. Możliwe jest dodanie kolejnych kluczy do datagramu, oczywiście przy odpowiednim rozszerzeniu rozmiaru bufora.