POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Marcin Jasiński

Sprawozdanie z zajęć laboratoryjnych Technologii Sieciowych

Protokoły tekstowe

14 grudnia 2018

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

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/klient_txt/klient_txt/klient_txt.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/serwer_txt/serwer_txt/serwer_txt.cpp

Protokół:

- połączeniowy,
- wszystkie dane przesyłane w postaci tekstowej (sekwencja znaków ASCII),
- każdy komunikat opatrzony znacznikiem czasu,
- nazwy pól o określonej długości: 2 znaki,
- struktura elementów nagłówka zdefiniowana jako klucz=wartość$
    - (przykład) Operacja=dodaj$
- wymagane pola:
    - pole operacji – „OP”,
    - pole statusu – „ST”,
    - pole identyfikatora – „ID”.
- dodatkowe pola zdefiniowane przez programistę.

Funkcje oprogramowania:

- nawiązanie połączenia,
- uzgodnienie identyfikatora sesji,
- wykonywanie operacji matematycznych na dwóch argumentach:
    - „poteguj” – potęgowanie,
    - „logarytmuj” – logarytmowanie,
    - 2 inne, zdefiniowane przez programistę.
- przeglądanie historii wykonywanych obliczeń:
    - po stronie klienta:
        - poprzez podanie identyfikatora sesji,
        - poprzez podanie identyfikatora obliczeń.
    - po stronie serwera:
        - poprzez podanie identyfikatora sesji,
        - poprzez podanie identyfikatora obliczeń,
        - wyświetlenie wszystkich dotychczas wykonanych obliczeń.
- zakończenie połączenia.

Inne:

- gdy wartość wyniku wykracza poza zakres zmiennej, powinien zostać zwrócony kod błędu,
- każde obliczenia powinny posiadać unikalny identyfikator,
- identyfikator sesji powinien być przesyłany w trakcie komunikacji i powiązany z obliczeniami,

- odwołanie się do nieistniejących obliczeń lub obliczeń wykonanych przez innego użytkownika, powinno skutkować przesłaniem odpowiedniego statusu.

2. Opis protokołu (format komunikatu, zbiór komend i odpowiedzi).

Protokół tekstowy jest przeznaczony do bycia czytanym przez ludzkiego użytkownika, raczej niż maszynę, w przeciwieństwie do protokołów binarnych. Są różne metody kodowania danych, najpopularniejszą jest używanie kodów znaków ASCII. Typowymi przykładami protokołów tekstowych są FTP (*File Transfer Protocol*) i SMTP (*Simple Mail Transfer Protocol*).

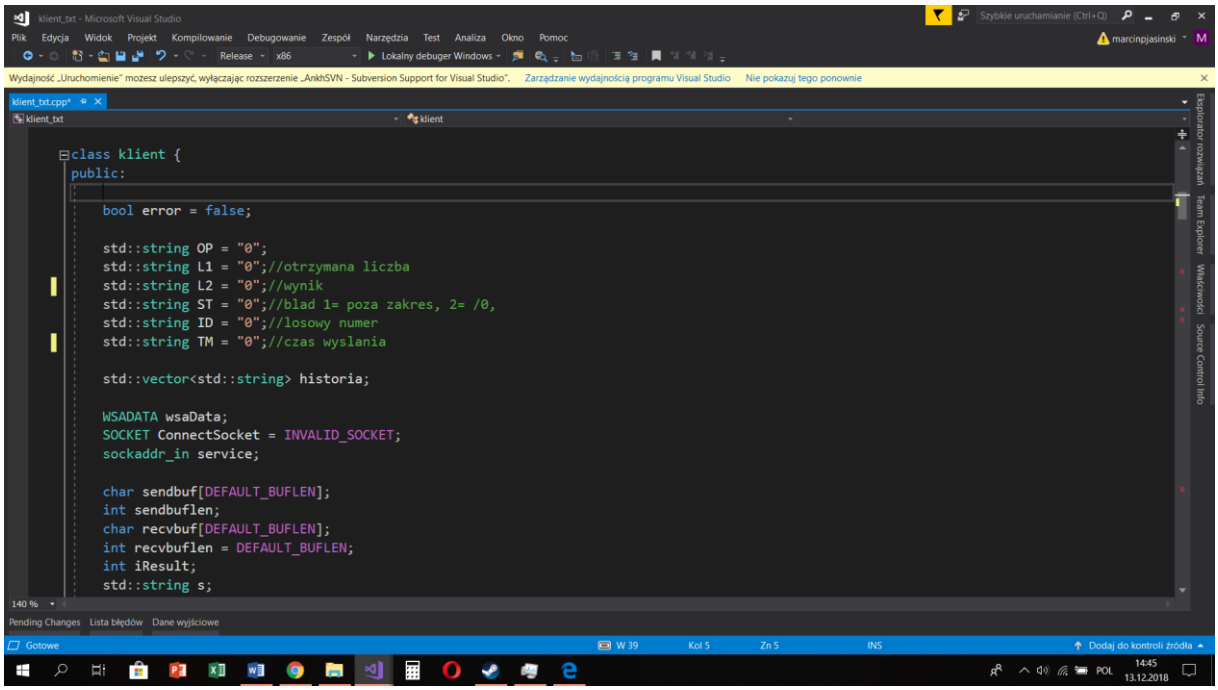
Charakterystyka protokołu TCP: Jest on niezawodnym protokołem warstwy transportowej, działającym w trybie klient-serwer. Serwer po uruchomieniu oczekuje na nawiązanie połączenia na określonym porcie przez klienta. Ten w końcu inicjuje połączenie, po czym komunikacja odbywa się w dwu kierunkach, czyli zarówno od nadawcy do odbiorcy jak i od odbiorcy do nadawcy. Podstawowe funkcje protokołu: realizacja dwukierunkowej transmisji połączeniowej, buforowanie danych, zarządzanie przepływem, zapewnienie niezawodności.

Moje programy są stworzone w języku c++, z użyciem biblioteki windsock2.

Faktyczna budowa komunikatu w programie:

- OP Operacje
  - potęgowanie,
  - logarytm $\log_{L1}L2$,
  - mnożenie,
  - dzielenie,
  - historia wysłanych danych
  - przyrównanie do danej liczby
- liczba 32 bitowa typu integer.
  - L1 liczba wysyłana przez klienta
  - L2 wynik pośredni odsyłany przez serwer, przechowywany w serwerze
- ST błąd
  - 1 - poza zakres,
  - 2 – nieprawidłowe podanie 0 (np. dzielenie przez 0),
- ID losowy numer
- TM czas wysłania to liczba sekund od 1 stycznia 1970
- Komunikat nie ma stałej długości.

Każde pole jest rozdzielone $, a opis pola z jego wartością =, czyli Klucz=wartość$

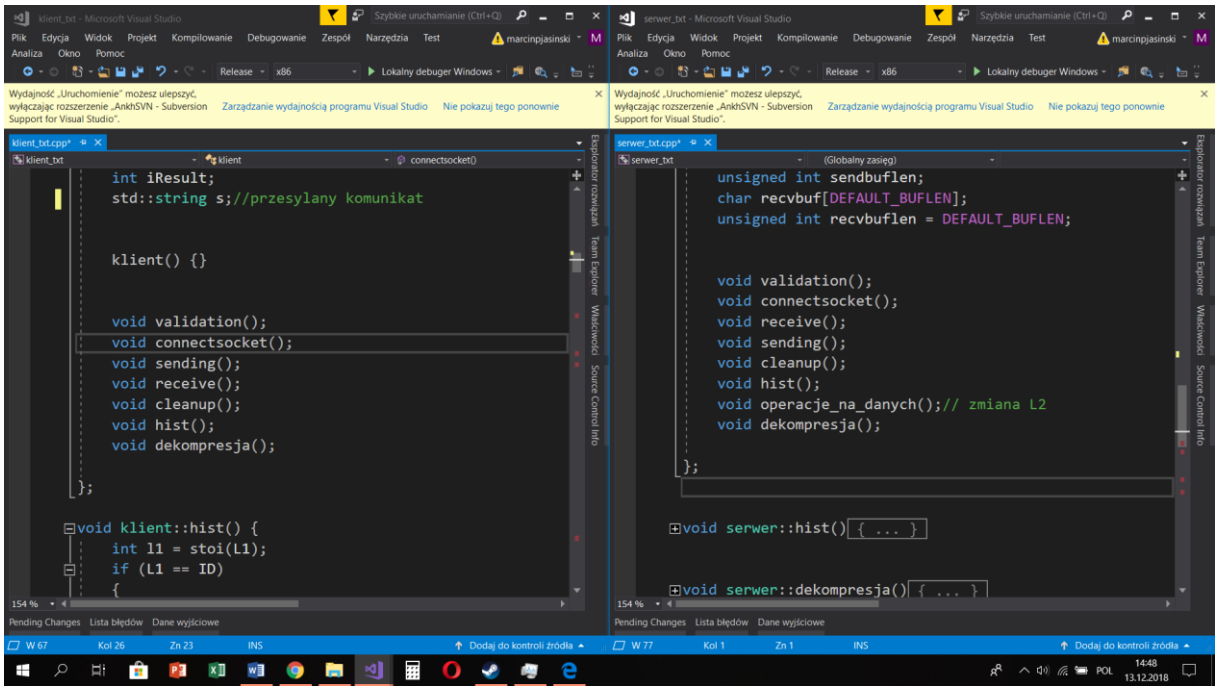


Zrzut ekranu nr 1 – pola komunikatu w programie

3. Aplikacja użytkownika oraz aplikacja serwera (zależnie od wariantu) w wersji źródłowej z komentarzami (w szczególności dotyczącymi fragmentów programów związanych z transmisją).

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/klient_txt/klient_txt/klient_txt.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/serwer_txt/serwer_txt/serwer_txt.cpp



Zrzut ekranu nr 2 – nazwy utworzonych funkcji klienta (po lewej) i serwera (po prawej)

Opis działania aplikacji klienta:

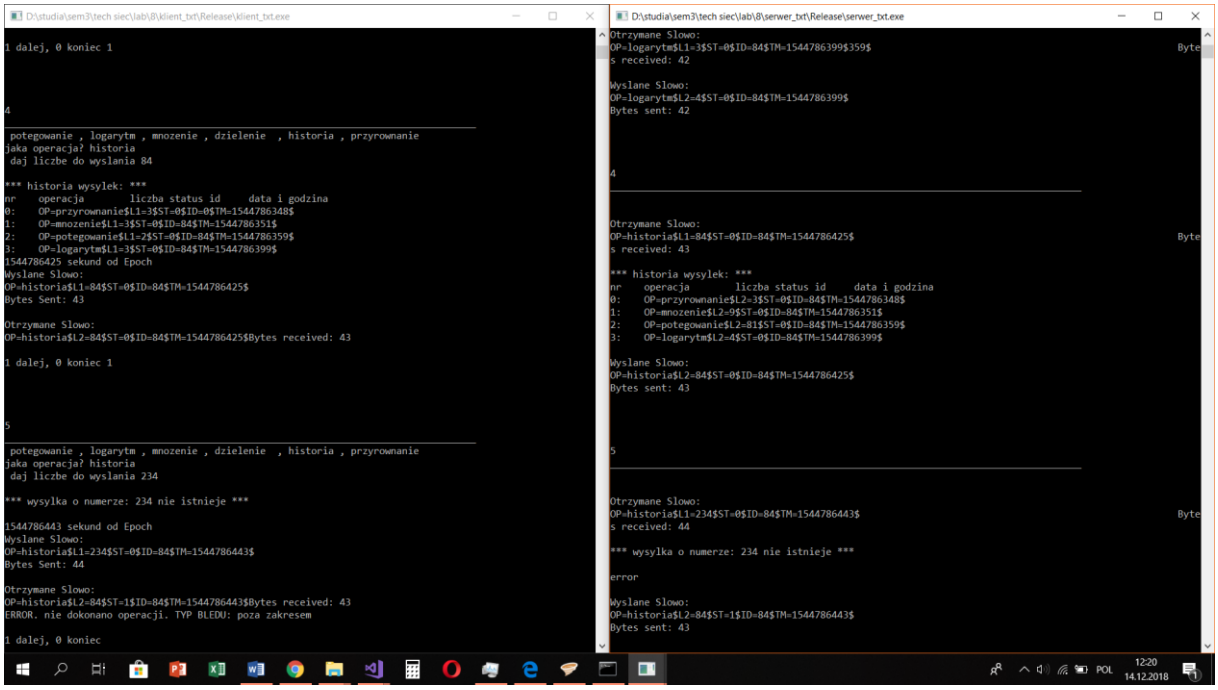
1. Klient po zainicjowaniu próbuje się połączyć z serwerem o danym adresie IP. (`klient::validation() i klient::connectsocket()`)
2. Po udanym połączeniu, serwer ustala identyfikator połączenia, a klient ustala początkową wartość serwera. (`main()`)
3. Klient podaje dane komunikatu: OP, L1.(`main()`)
4. Jeśli operacją jest historia, (`hist()`) na konsoli wyświetlony zostaje:
    - Przesłany komunikat o danym indeksie, jeśli podano liczbę odpowiadającą mu,
    - Wszystkie komunikaty wysłane przez klienta, jeśli podano indeks połączenia,
    - Informację o błędzie, jeśli podana liczba nie odpowiada poprzednim przypadkom.
5. Klient wysyła serwerowi komunikat (`klient::sending()`)
6. Klient odbiera odpowiedź serwera, wyświetla wyniki częściowe i ewentualnie kody statusu, informacje o błędach.(`klient::receive() i main()`)
7. klient się pyta, czy będzie więcej komunikatów. Jeśli tak, to powrót do punktu trzeciego `(main())`
8. Jeśli nie ma więcej komunikatów, połączenie jest zamykane (`klient::cleanup()`)

Opis działania aplikacji serwera:

1. Po udanych inicjalizacji i połączeniu, serwer ustala identyfikator połączenia (`serwer::validation()i serwer::connectsocket()`)

2. Serwer, pyta się czy będzie chciał się pytać o historię przesyłek, niezależnie od otrzymanego komunikatu.(wywoływanie `hist()` przy każdej przesyłce.)
3. serwer odbiera komunikat od klienta (`serwer::receive()`)
4. serwer operuje na danych. Zmienia wartość L2, jeśli nie pojawia się błąd. W przeciwnym wypadku, Ustawia status na odpowiednią wartość (ST), a L2 się nie zmienia. (`serwer::operacje na danych()`)
5. Jeśli operacją jest historia, (`hist()`) na konsoli wyświetlony zostaje:
    - Przesłany komunikat o danym indeksie, jeśli podano liczbę odpowiadającą mu,
    - Wszystkie komunikaty wysłane przez serwer, jeśli podano indeks połączenia,
    - Informację o błędzie, jeśli podana liczba nie odpowiada poprzednim przypadkom.
6. serwer wysyła odpowiedź (`serwer::sending()`)
7. Jeśli nie ma więcej komunikatów, połączenie jest zamykane. (`serwer::cleanup()`)

Po każdym wysłaniu komunikatów, komunikat jest przechowywany w `vector<string>historia`, na wypadek potrzeby późniejszego wyświetlenia.

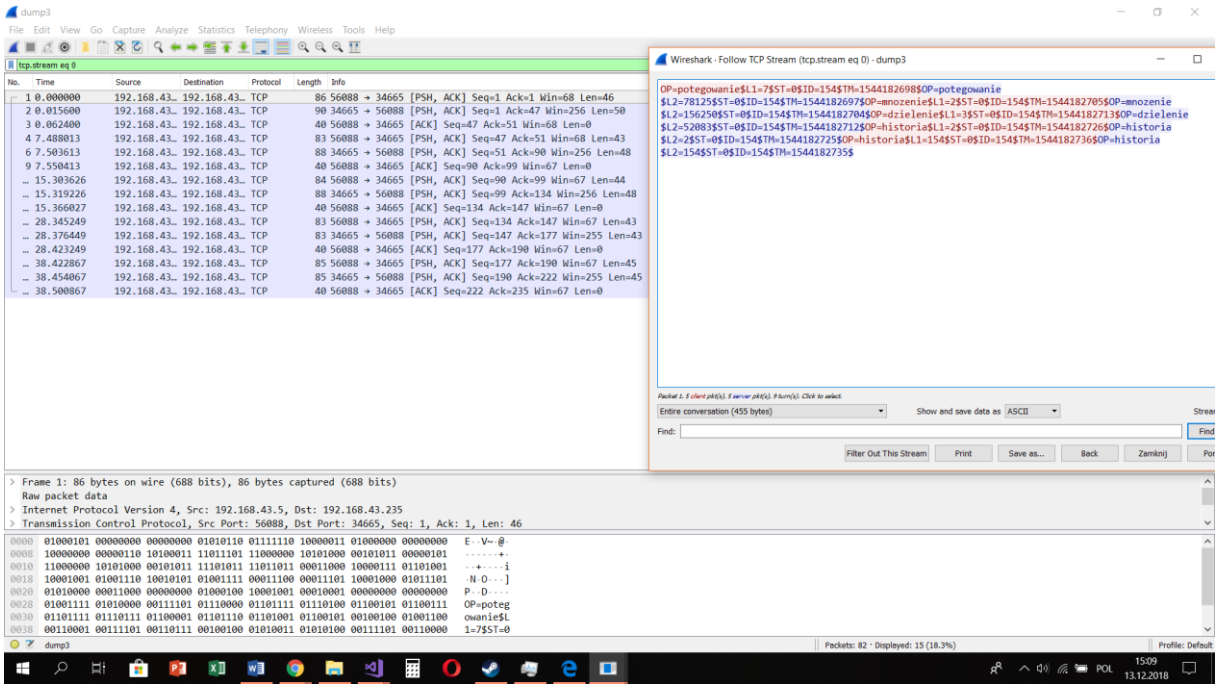


Zrzut ekranu nr 3 – przykład działania historii w programie.

Na zrzucie ekranu nr 3 klient jest po lewej, a serwer jest po prawej. Numer nad kreską oznacza numer operacji. W operacji nr 4 podano operację historia i liczbę 84, która jest identyfikatorem połączenia, więc wyświetlono historię wysyłek po obu stronach. W operacji nr 5 podano liczbę nie odpowiadającą żadnej wysyłce, więc zwrócono błąd i nadano statusowi wartość 1 (poza zakresem).

4. Przebieg przykładowej sesji komunikacyjnej – opis słowny oraz obraz sesji zarejestrowany przez program Wireshark, wraz ze stosownymi objaśnieniami.

Przed początkiem tej rejestracji, przyrównano L2 w serwerze do 5.



Zrzut ekranu nr 4 - prośby klienta o działanie na czerwono i odpowiedzi serwera na niebiesko

Tabela 1: Podział pierwszego przesyłanego komunikatu na zrzucie ekranu nr 3:

Część komunikatu	Zawartość	interpretacja
pole operacji OP	potegowanie	Operacja potęgowania
pole liczby L1	7	5 ma być podniesione do 7 potęgi
pole statusu ST	0	Operacja przebiegła bez błędów
pole identyfikatora	154	Identyfikator sesji wynosi 154
Znacznik czasu TM	1544182698	Ilość sekund od 1 stycznia 1970

Tabela 2: Podział drugiego komunikatu na zrzucie ekranu nr 3:

Część komunikatu	zawartość	interpretacja
pole operacji OP	potegowanie	Operacja potęgowania
pole liczby L1	78125	Wynik pośredni 5^7 = 78125

pole statusu ST	0	Operacja przebiegła bez błędów
pole identyfikatora	154	Identyfikator sesji wynosi 154
Znacznik czasu TM	1544182698	Ilość sekund od 1 stycznia 1970

5. Odpowiedzi na pytania postawione w sekcji „Zadania szczegółowe”

1. **Przygotuj implementacje protokołu komunikacyjnego, aplikacji klienckiej oraz aplikacji serwerowej w dowolnym języku wysokiego poziomu.**

Obie aplikacje zostały wykonane w języku c++. Kody źródłowe znajdują się na repozytorium o podanych linkach:

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/klient_txt/klient_txt/klient_txt.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/serwer_txt/serwer_txt/serwer_txt.cpp

2. **Przetestuj połączenie pomiędzy programami, rejestrując całość transmisji. Przeanalizuj przechwycone dane. Czy przesłane dane są w pełni tekstowe?**

Całość transmisji na zrzucie ekranu nr 4.

Nie występuje potrzeba dopełnienia do bajtów, bo przesyłane dane składają się z bajtów, co oznacza, że przesyłane dane są w pełni tekstowe.

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

W stworzonym programie rozmiar komunikatu zależy od wartości przesyłanych danych, bo pola komunikatu mogą zawierać różną ilość bajtów. Rozmiar się zmieni gdy dane się zmienią, lub rozbuduje się protokół dodając nowe dane. W programie można łatwo rozbudować protokół. Starczy dodać nowe pola do przesyłanej tablicy znaków.