POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY

Kacper Kociubiński
Jakub Kolasiński

Sprawozdanie

PROJEKT APLIKACJI WYKORZYSTUJĄCEJ AUTORSKI PROTOKÓŁ KOMUNIKACYJNY

Wstęp

Projekt przedstawia implementację porównywarki cen walut składającej się z aplikacji klienta oraz serwera. Został on oparty o asynchroniczne wzorce programowania oraz autorski użytkowy protokół komunikacji. Program jest gotową aplikacją pozwalającą użytkownikowi na śledzenie aktualnych uśrednianych cen walut oraz porównywanie ofert kantorów internetowych. Rozkłada swoje działanie pomiedzy dwie aplikacje klienta i serwera.

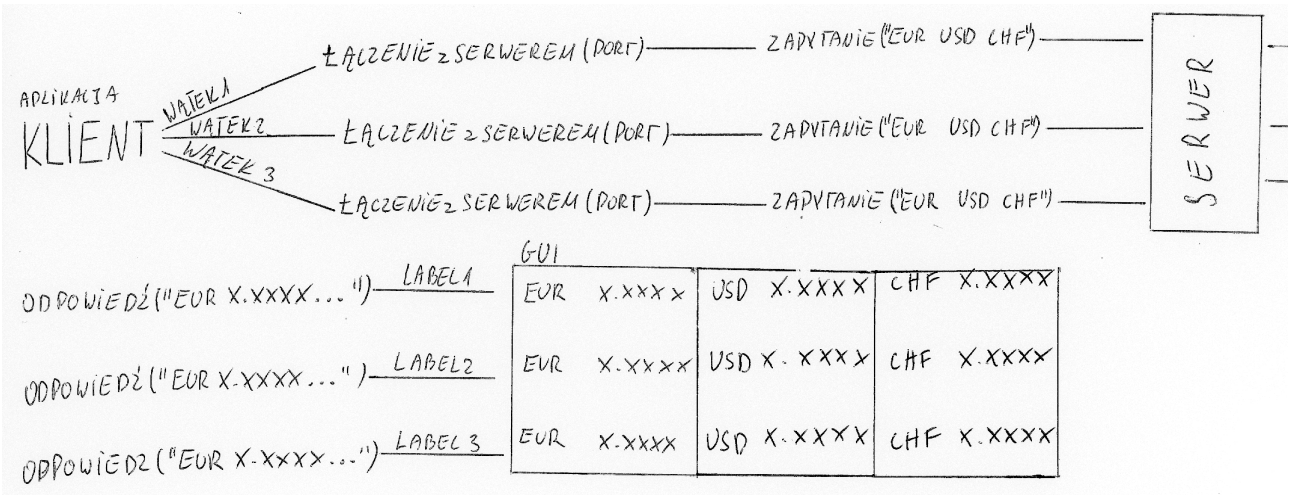
Założenia

Implementacja asynchronicznej architektury klient - serwer pozwalającej na pobieranie danych ze stron internetowych i ich przetwarzanie. Poprawna komunikacja pomiędzy aplikacjami oparta o autorski protokół. Porównanie szybkośći tak działającej aplikacji w stosunku do podobnego programu, który nie wykorzystuje w pełni wzorców programowania asynchronicznego.

Budowa aplikacji

Klient

Aplikacja klienta jest programem spełniającym zadanie informowania użytkownika na żądanie o kursach wybranych przez niego walut. Posiada ona interfejs graficzny przystosowany do obsługiwania euro, dolarów amerykańskich i franków szwajcarskich oraz stron kantorów internetowych walutomat.pl, internetowykantor.pl, kantoria.pl. Klient sam definiuje jakie waluty go interesują. W momencie żądania przez użytkownika wyświetlenia kursów zostaje uruchomiony mechanizm oparty o zadnia, który w asynchroniczny sposób będzie łączy się z aplikacją serwera. Działanie rozpoczynają trzy wątki wywołujące funkcję ClientTask(numer port) odpowiedzialną za połączenie się z serwerem dedykowanym do obsługi danego kantoru internetowego. Każdy z nich wykorzystuje kolejno odpowiadające im porty 2048, 2049, 2050. Wysłanie zapytania jest realizowane za pomocą autorskiego protokołu. Oczekiwanie na odpowiedź jest realizowane asynchronicznie. Po uzyskaniu informacji zwrotnej wątki kolejno parsują otrzymane dane według schematu protokołu i przekazują informację bezpośrednio do etykiety przeznaczonej dla konkretnej waluty oraz kantoru. Wszelkie dane zostają wyświetlone użytkownikowi na ramach okna interfejsu graficznego. Aplikacja klienta pozwala na wysłanie wielu zapytań, które będą realizowane po kolei.



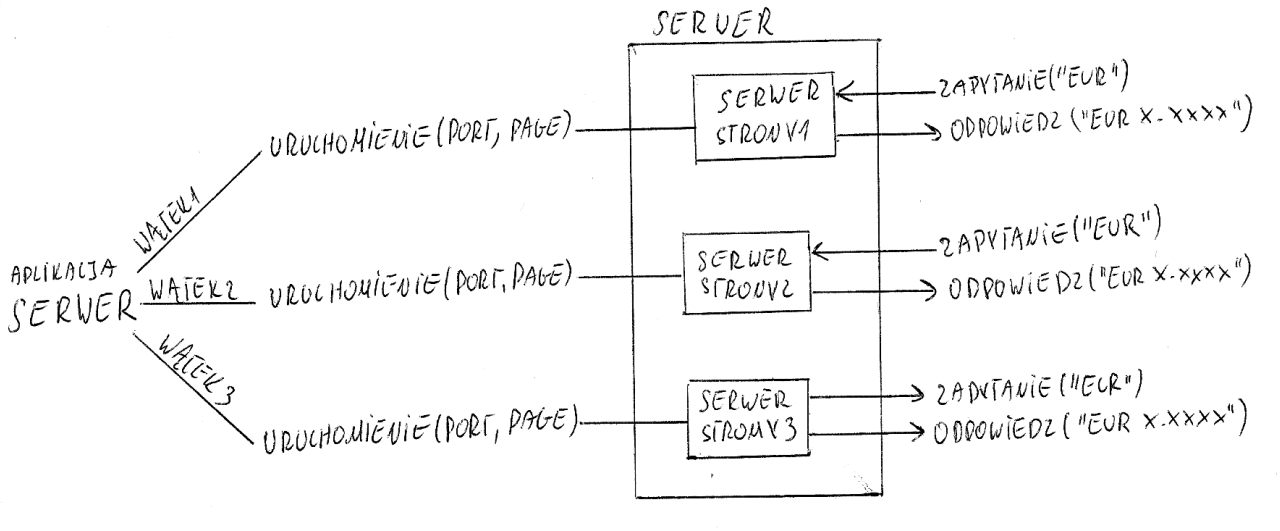
Schemat 1. Aplikacja klienta

Serwer

Aplikacja serwera jest programem obsługującym klienta. Nie posiada interfejsu graficznego. Swoje działanie rozpoczyna od trzech zadań mających na celu uruchomienie trzech serwerów na portach 2048, 2049, 2050. Zadania w argumentach oprócz numeru portu posiadają strukturę przekazującą im link do strony internetowej kantoru oraz format węzła Htlm serverTask(numer portu, Page{link, węzeł htlm}). Każdy uruchomiony serwer jest w stanie obsługiwać wiele zapytań i wielu klientów w osobnych wątkach. Ich głównym zadaniem jest odpowiedź na otrzymane zapytanie w protokole. Wykorzystują do tego funkcję asynchroniczną GetRatesAsync pozwalające na pobranie odpowiedniej strony internetowej, dotarcie do węzła przechowującego wartość uśrednionego kursu walut oraz parsowanie informacji do oczekiwanej struktury protokołu.

Serwery są dedykowane dla określonych stron:

- port 2048 - walutomat.pl,
- port 2049 - internerowykantor.pl,
- port 2050 – kantoria.pl.



Schemat 2. Aplikacja serwera

Protokołu komunikacyjny

Autorski protokół komunikacyjny jest przeznaczony do szybkiej obsługi wymiany informacji o kursach walut pomiędzy klientem a serwerem. Posiada dwa główne zadania i przez to dwie formy występowania, które są uzależnione od ilość kursów, o jakie pyta klient. Zapytania są kierowane od klienta i mają następującą formę:

- żądanie informacji o kursie euro

EUR

- żądanie informacji o kursie euro, dolara amerykańskiego i franka szwajcarskiego

EUR	USD	CHF

Odpowiedzi serwera wyglądają następująco:

- odpowiedź na zapytanie o kurs euro

EUR 4,2924

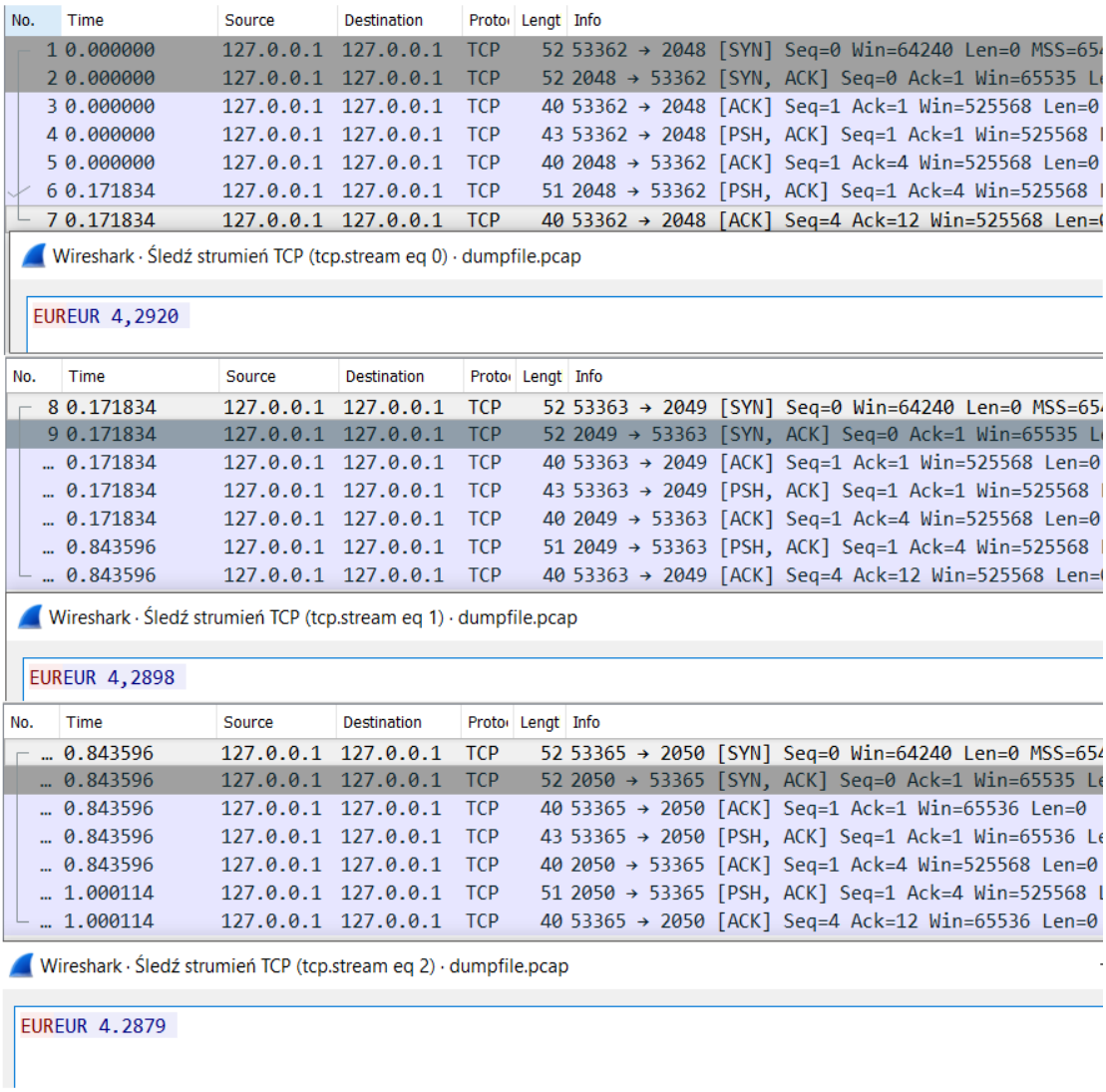
- odpowiedź na zapytanie o kurs euro, dolara amerykańskiego i franka szwajcarskiego

EUR 4,2924	USD 3,7743	CHF 3,7902

Rezultaty

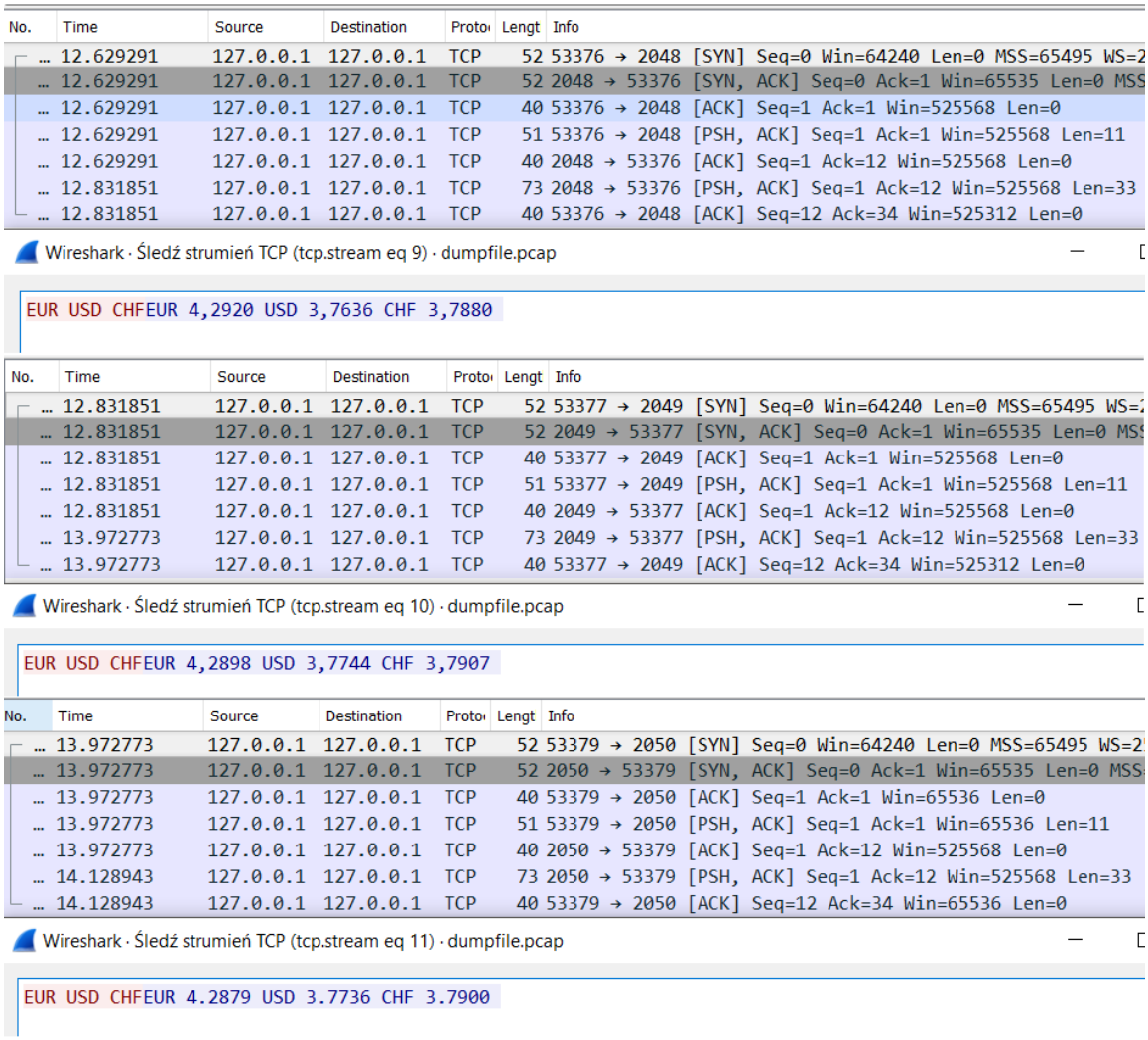
Komunikacja i protokół komunikacyjny

Komunikacja między serwerem i klientem przebiega poprawnie. Aplikacje nawiązują miedzy sobą połączenie i wymieniają informacje. Nie zauważyliśmy żadnych, nieprzewidywalnych zerwał połączenia. Wysyłane zapytania oraz odpowiedzi mają format przewidzianego przez nas protokołu komunikacyjnego. Badania połączenia dokonaliśmy za pomocą programy RawCap - sniffera wymienianych pakietów w obrębie portów jednej kary sieciowej oraz WireShark - analizatora pakietów.



Rysunek 1. Wymiana pakietów pomiędzy serwerem i klientem dla jednej waluty.

Na *Rysunku 1* został przedstawiony przebieg komunikacji będący wynikiem żądania przez użytkownika informacji na temat aktualnego kursu waluty euro. Zostały wysłane trzy zapytania o treści EUR do trzech serwerów na kolejnych ortach 2048, 2049, 2050. Serwery po przetworzeniu zapytani i uzyskaniu informacji zwrotnej ze stron internetowych kantorów, odpowiadają następującą informacją EUR X,XXXX - gdzie X,XXXX jest aktualną ceną waluty.



Rysunek 2. Wymiana pakietów pomiędzy serwerem i klientem dla wielu walut.

Rysunek 2 przedstawia przebieg komunikacji dla żądania uzyskania informacji o cenie walut euro, dolar amerykański oraz frank szwajcarski przez klienta. Został tu wykorzystany format protokółu przeznaczony do tworzenia zapytań i odpowiedzi, gdy pytamy o wiele kursów.

Aplikacja synchroniczna a asynchroniczna

W celu lepszego zobrazowania jakie możliwości dają nam wzorce programowania asynchronicznego, porównaliśmy działanie naszego programu z podobną aplikacją, która jednak nie wykorzystuje w pełni współbieżności.

Różnice wynikają z budowy serwera:

- funkcja GetRatesAsnyc jest funkcją synchroniczną, każde parsowanie danych i przeszukiwanie węzła Html zostaje wykonane przez ten sam wątek,
- stronynie są pobierane w sposób asynchroniczny.

Zmiany w działaniu klienta:

- tworzenie połączeń z kolejnymi portami nie opiera się o zadania, a jest wykonywane synchronicznie,
- trzej utworzeni klienci działają i komunikują się z serwerem za pomocą jednego wątku.

Badaliśmy czas po stronie aplikacji klienta, jaki program asynchroniczny i synchroniczny potrzebuje na połączenie się z trzema serwerami, wysłanie zapytania oraz otrzymania poprawnej odpowiedzi. Zarówno dla jeden waluty, jak i trzech dokonaliśmy dwudziestu pomiarów oraz wyniki uśredniliśmy.

1 waluta async [s]	**1 waluta sync [s]**	**3 waluty async [s]**	**3 waluty sync [s]**
00.5264667	01.5112153	00.5167988	01.2846927
00.4798099	00.9985951	00.5372456	00.9360506
00.5297105	01.2945336	00.6643854	01.6155206
00.5799948	00.9040229	00.5853862	01.1527982
00.5357348	01.0405352	00.5626563	00.9046870
00.5782099	00.9940645	00.7214074	01.0465781
00.5325751	00.9092547	00.7775136	00.9485867
00.6457338	00.8275797	00.5522733	00.9582657
00.5510206	01.1309130	00.5559027	00.9204358
00.5232256	01.0586346	00.6873250	00.9003224
00.5493925	01.2545250	00.5501692	01.0304138
00.5448252	00.9132911	00.7097337	00.8392246
00.5405190	00.9627441	00.5902608	00.9271163
00.5212361	01.2113685	00.5103249	01.0249789
00.5510237	01.0455679	00.5888382	01.7211415
00.5200442	01.4829641	00.7418178	01.0371377
00.5095833	00.9171417	00.5322753	00.9352138
00.5379839	00.9862680	00.5264979	00.9536442
00.5526960	00.9544462	00.5428467	00.9535309
00.5052294	01.1346547	00.5639632	01.0473437
Średnia			
00.5407508	**01.0766160**	**00.6008811**	**01.0568842**
Uzyskane przyśpieszenie			
49,80%		**43,20%**	

Uzyskaliśmy optymalne przyśpieszenie, które wynika głównie ze zrównoleglenia pobierania stron internetowych. Jest to operacja, która trwa najdłużej. W aplikacji synchronicznej strony ładowane są po kolei, przez co czas poświęcony na wszystkie możemy określić wzorem:

$$\Delta t = t_1 + t_2 + t_3$$

gdzie:

- t_1 , t_2 , t_3 – czasy pobierania kolejnych stron internetowych,
- Δt – czas całkowity.

Natomiast przy zrównolegleniu trzy strony internetowe będą, pobieramy się obok siebie. O czasie zadecyduje wtedy funkcja, która będzie ściągała stronę najdłużej, ponieważ w ty momencie pozostałe funkcje zdąża wykonać swoją operację.

$$\Delta t = t_{max}$$

gdzie:

- t_{max} – najdłuższy czasy pobierania kolejnych stron internetowych,
- Δt – czas całkowity.

Wnioski

Implementacja aplikacji w architekturze klient - serwer oparta o asynchroniczne wzorce programowe (w tym przypadku zadania) przynosi wymierne efekty w postaci przyspieszenia działania. Udostępnia ona również możliwość obsługi wielu użytkowników jednocześnie w krótkim czasie, co jest obecnie bardzo pożądaną cechą.

Główną klasą użytą w implementacji asynchronicznej aplikacji jest klasa Task, a wraz z nią użyte zostały instrukcje await i async które pozwalają tworzyć metody asynchroniczne oraz sterować zachowaniem obiektów Task oraz wątków. Wzorzec programowy oparty na zadaniach jest wygodny w użyciu oraz przejrzysty, przez co tworzenie asychronicznych aplikacji jest nie bardziej skomplikowane od tworzenia wersji synchronicznych.