Programowanie sieciowe

Program końcowy 2024/2025 (zima) - NS **dr hab. inż. Radosław Wajman**

Serwer czasu

Cel ćwiczenia:
Celem ćwiczenia jest implementacja usługi Serwera Czasu (SC) wraz z przykładowymi modułami serwera i klienta. Taka usługa często przydaje się do wzajemnej czasowej synchronizacji różnych urządzeń pomiarowych o różnej częstotliwości akwizycji. Hipotetycznie ujmując, taka zaprojektowana usługa mogłaby działać w oparciu o lokalną sieć komputerową LAN lub WAN. Na jednym z komputerów zaprojektowana jest aplikacja SC. Jej zadaniem byłoby np. udostępnianie klientom lokalnego czasu tego komputera. Każdy z klientów, zanim przystąpiłby do rejestrowania pomiarów z podłączonych do niego urządzeń, zobligowany będzie do wysłania do serwera żądania o podanie jego lokalnego czasu systemowego. Na podstawie tej wartości oraz swojego czasu lokalnego i czasu transmisji danych, wyznaczana jest różnica pomiędzy czasem klienta, a czasem serwera, która później może posłużyć do wyliczenia znacznika czasowego. Ten znacznik czasowy powinien być dodawany do każdego komunikatu rejestrującego dane.
W trakcie realizacji zadania studenci opracują dwa algorytmy do znajdywania adresu IP serwera czasu nasłuchującego na zadanym porcie oraz automatycznego utrzymywania i wznawiania połączenia z serwerem.

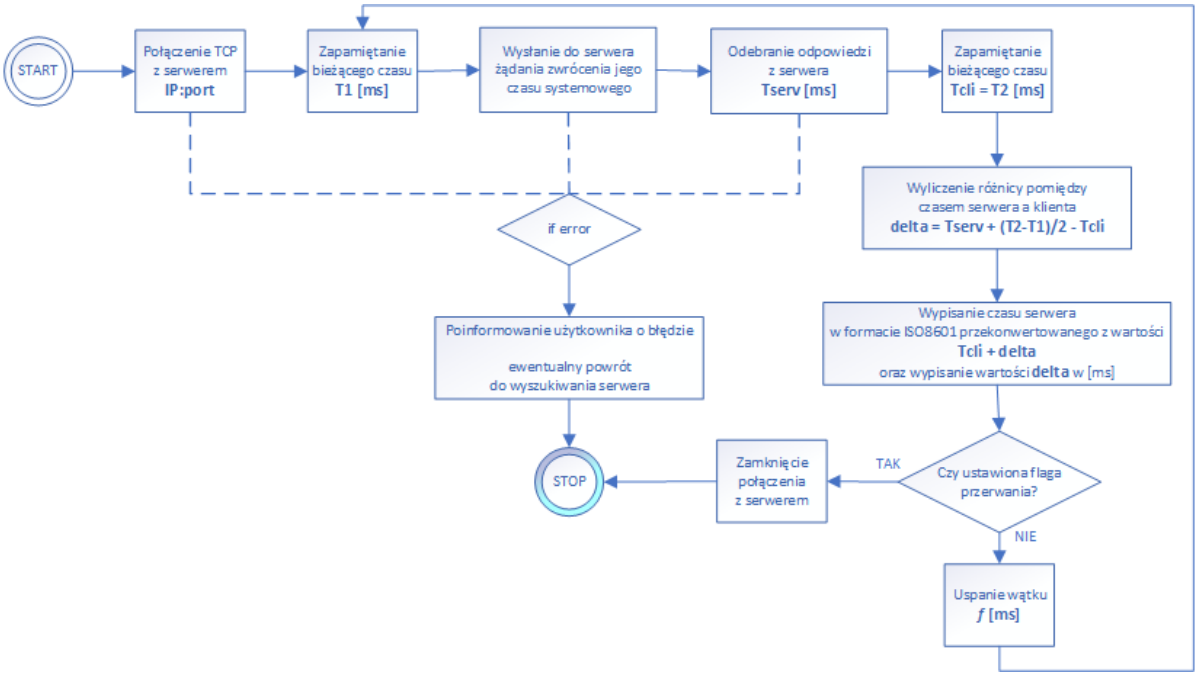
Polecenie ćwiczeniowe:
Do zaliczenia ćwiczenia wymagane jest napisanie programu klienta oraz serwera czasu, których funkcjonalność zdefiniowano poniżej. Wybór platformy programistycznej należy do studenta.

moduł klienta:
1. Umożliwia podłączanie do modułu serwera,
2. Wysyła dane przy użyciu protokołu TCP,
3. Nie zna adresu IP serwera oraz numeru portu komunikacji, w tym celu wysyła zapytanie w trybie *multicastu* o treści np. DISCOVERY na porcie np. 7,
4. Po otrzymaniu ofert serwerów wraz z numerem portu oferowanej komunikacji TCP:
    a. wypisuje adresy IP oraz numery portów serwerów, które odpowiedziały na wołanie klienta pomijając adres typu localhost (127.0.0.1),
    b. daje użytkownikowi możliwość wyboru serwera i natychmiast inicjalizuje połączenie,
    c. wyróżnia serwer, z którym był ostatnio połączony jako domyślny,
5. W przypadku nieudanego podłączenia TCP z serwerem lub jego utraty wypisuje stosowny komunikat i niezwłocznie przechodzi do realizacji punktu 3 lub 4b,
6. Aktualizacja listy dostępnych serwerów powinna odbywać się na bieżąco z częstotliwością co 10 sekund, ale tylko w przypadku braku aktywnego połączenia z serwerem TCP, o którym mowa w punkcie 5),
7. Po otrzymaniu od użytkownika rozkazu połączenia z serwerem TCP:
    a. pyta o częstość wysyłania zapytań do serwera $Ts \in (10 - 1000)ms$,
    b. uruchamia wątek komunikacji z serwerem i pracuje zgodnie z algorytmem przedstawionym na rys. 1 na następnej stronie,

moduł serwera:
8. Od chwili uruchomienia ma realizować ciągłe nasłuchiwanie TCP na wszystkich rutowalnych interfejsach sieciowych dostępnych w komputerze na losowo dobranym numerze portu (każdy interfejs to inny port),
9. Ma wyświetlać adres(y) IP i numer(y) porty(ów), na którym(ch) nasłuchuje,
   Aby móc efektywnie przetestować funkcjonalność wielu kart sieciowych, można skorzystać z wirtualizacji i poza siecią podstawową utworzyć oddzielne podsieci z komputerami wirtualnymi (zobacz podpowiedź),
10. Gdy klient (TCP) podłączy się do serwera, uruchamia dla niego oddzielny wątek i odbiera od niego żądanie, następnie pobiera swój bieżący czas systemowy w [*ms*] i niezwłocznie wysyła tę wartość do klienta,
11. Gdy serwer otrzyma informacje, że klient (TCP) rozłączył się, to również zamyka połączenie i kończy wątek klienta,
12. Serwer dodatkowo zaraz po uruchomieniu ma rozpocząć nasłuchiwanie komunikatów DISCOVER na porcie 7 z użyciem protokołu UDP i odpowiadać komunikatem OFFER ADDRESS PORT,

a. dla każdego dostępnego interfejsu sieciowego (o którym mowa w pkt. 8), wątek UDP wysyła oddzielne pakiety OFFER,
b. nasłuchiwanie w trybie multicastu nigdy nie ustaje,

13. Serwer prowadzi na bieżąco statystyki dotyczące aktualnie podłączonych/rozłączonych klientów i kolejnych odebranych żądań DISCOVER i wyświetla je użytkownikowi podając za każdym razem adresy IP klientów oraz IP serwera.



Rys. 1 Algorytm wątku komunikacji klienta z serwerem

Należy zwrócić uwagę, że testowanie programu najlepiej powinno odbywać się pomiędzy dwiema różnymi maszynami. Najlepiej, jeśli obie maszyny będą w różnych podsieciach, tzn. kiedy ruch sieciowy będzie odbywał się przynajmniej przez jeden router. Można to zrealizować na jednym komputerze korzystając z wirtualizacji i ustanowić transmisję pomiędzy systemem gościa i gospodarza.

Kryteria oceny:

1. Program powinien spełniać wszystkie kryteria opisane powyżej, w tym:
   a. kolorem czarnym obowiązkowe na ocenę 3,
   b. kolorem zielonym na 4,
   c. kolorem czerwonym na 5,

   Funkcjonalności określone na ocenę powyżej 3 nie są zamienne,
2. Samodzielność realizacji programu
   a. niedozwolone jest korzystanie z pomocy innych studentów,
   b. dozwolone jest korzystanie z własnych notatek oraz zasobów z internetu,
3. Ukończony i działający program należy wysłać za pomocą modułu zadania na stronie przedmiotu WIKAMP (archiwum <20MB), a następnie zaprezentować go osobiście (lub poprzez platformę MsTeams z włączoną kamerą oraz z opcją udostępnienia pulpitu) będąc jednocześnie przygotowanym z zakresu materiału prezentowanego na wykładach,
4. Ocenie podlegać będzie:
   a. wiedza teoretyczna – wykładowa,
   b. plik README umieszczony w głównym katalogu projektu z informacją o środowisku programistycznym użytym do realizacji zadania oraz strukturze folderów projektów, jeśli jest ona złożona),
   c. samodzielność realizacji zadania (jeżeli w trakcie projektowania zostały włączone fragmenty kodu zaczerpnięte ze źródeł internetowych, to w pliku README należy umieścić stosowne referencje do tych źródeł oraz wskazać zakres implementacji – to z pewnością nie obniży oceny, a jedynie pozwoli mi wyeliminować możliwość ew. niesamodzielności pracy),

d. oryginalność implementacji własnego protokołu komunikacyjnego klient – serwer opartego na TCP i UDP i standardzie JSON,
e. spójność kodu i optymalizacja algorytmów,
f. zużycie czasu procesora, liczba wątków, ich synchronizacja oraz kończenie, złożoność algorytmów,
g. odporność na sytuacje wyjątkowe (zrywanie połączeń, błędne dane od użytkownika), zaimplementowana procedura obsługi błędów i sytuacji wyjątkowych,
h. synchronizacja wątków,
i. czytelność kodu.

Powodzenia!