Programowanie sieciowe

Program końcowy 2023/2024 (lato) **dr hab. inż. Radosław Wajman**

Usługa serwera pośredniczącego w transmisji komunikatów

Cel ćwiczenia:
Celem ćwiczenia jest implementacja usługi Serwera Pośredniczącego (SP) wraz z przykładowymi modułami klientów. Zadaniem modułu serwera będzie przekazywanie komunikatów pomiędzy różnymi modułami klientów przy spełnieniu zdefiniowanych warunków komunikacji oraz formatu komunikatów. Komunikacja będzie identyfikowana poprzez tzw. tematy i dzielić będzie moduły klientów pod względem pełnionej funkcji na producenta i subskrybentów.

Polecenie ćwiczeniowe:
Do zaliczenia ćwiczenia wymagane jest napisanie programu Serwera Pośredniczącego (SP) oraz API Klienta (API) tj. interfejsu programistycznego (pakietu/biblioteki) dołączanego do oprogramowania. Oba moduły (tj. serwer SP i API) mogą być zrealizowane w dowolnych technologiach. Komunikacja oparta na bezpośredniej implementacji protokołu TCP (nie WebSocket!). Aplikacje (serwer i klient) będą realizować następujące funkcjonalności:

Aplikacja Serwera (SP):

- zaraz po uruchomieniu program:
  - wczytuje z pliku np. `config.json` podstawową konfigurację pracy aplikacji w tym:
    - nazwę serwera: ServerID,
    - zasięg widoczności gniazda nasłuchującego (0.0.0.0, 127.0.0.1 lub inne): ListenAddresses,
    - port nasłuchiwania: ListenPort,
    - wartość w sekundach limitu czasu TIME_OUT wystąpienia zdarzenia na gniazdach nasłuchującym oraz serwera: TimeOut
    - listę adresów IP w notacji CIDR do weryfikacji, czy połączenia klientów pochodzą z uprawnionych sieci komputerowych i jeśli wartość any, to nie jest realizowana weryfikacja IP: AllowedIPAdresses,
    - wartość w bajtach limitu maksymalnej wielkości komunikatu: SizeLimit,

    Przykładowa zawartość pliku `config.json` została umieszczona na końcu tej instrukcji.
  - tworzy następujące zasoby:
    - Listę Tematów (LT),
    - Kolejkę Komunikatów Odebranych (KKO),
    - Kolejkę Komunikatów do Wysłania (KKW),
  - uruchamia wątek komunikacji serwera,
  - uruchamia wątek monitorujący, który zarządza tematami komunikacji do pośredniczenia,
  - uruchamia wątek interfejsu użytkownika,
- w wątku komunikacji serwera:
  - buduje gniazdo „*listenera*" do nasłuchiwania na zdefiniowanym interfejsie sieciowym i porcie protokołu TCP (czytaj punkt dot. wczytywania konfiguracji aplikacji),
  - ustawia dla gniazda „listenera" limit czasu TIME_OUT oczekiwania na żądanie połączenia klienta,
  - uruchamia nasłuchiwanie żądań połączenia, co zaimplementowane jest wewnątrz pętli, która kończy się spełnieniem warunku ustawienia flagi STOP_SERVER lub przechwyceniem wyjątku błędnej pracy gniazda z wyłączeniem TIME_OUT,
  - obsługuje żądanie podłączenia klienta i sprawdza, czy jego adres IP znajduje się w liście AllowedIPAdresses (chyba, że ustawiono wartość any) i jeżeli nie, to:
    - rozłącza połączenie lub jeżeli tak, to,
    - uruchamia wątek obsługi podłączonego klienta przekazując do niego utworzone gniazdo klienta (nie chodzi o „listener", ale powrót funkcji accept) do komunikacji z klientem,
  - ustawienie flagi STOP_SERVER kończy słuchanie i kończy wątek,
- w wątku obsługi podłączonego klienta:

- otrzymuje jako parametr gniazdo klienta i ustawia dla tego gniazda limit czasu TIME_OUT oczekiwania na przysłanie żądanie,
- realizuje oczekiwanie na komunikat żądania, co zaimplementowane jest wewnątrz pętli kończonej flagą STOP_CLIENT lub przechwyconym wyjątkiem pracy gniazda klienta z wyłączeniem tych dotyczących TIME_OUT,
- po otrzymaniu nowej wiadomości umieszcza ją w KKO wraz z referencją do posiadanego gniazda klienta,
- rozłączenie klienta powinno zakończyć wątek ale dodatkowo uwarunkowane jest realizacją procedur wymienionych w ramce: ROZŁĄCZANIE KLIENTÓW umieszczonej w opisie funkcji zarządzania komunikatem w sekcji dotyczącej typu komunikatu **withdraw**,

- rejestruje callback uruchamiany w chwili zmiany zawartości KKW i na bieżąco realizuje wysyłanie komunikatów z KKW (parametry każdego obiektu w KKW: referencja do komunikatu, tablica gniazd odbiorców komunikatu), a następnie usuwa wysłane obiekty z KKW,

UWAGA! dopuszczalne jest wykonanie tej funkcjonalności inną techniką niż callback, o ile wybrane narzędzia programistyczne nie wspierają obsługi zdarzeń zmian zawartości list,

- w wątku monitorującym realizuje nadzór nad obsługą komunikatów działając w pętli i uruchamiając sekwencyjnie następujące funkcje:
  - jeśli KKO jest puste, usypia się na 1ms,
  - pobranie kolejnego komunikatu (KOM) z KKO i jego walidacja w oparciu o specyficzny format tj.:

```
{
  "type": "register/withdraw/reject/acknowledge/message/file/config/status",
  "id": "SenderID",
  "topic": "lunch call",
  "mode": "producer/subscriber",
  "timestamp": "2024-02-15T00:00:00.000Z",
  "payload": {
    "....": "...."
  }
}
```

przy czym pola:

type: zawiera typ komunikatu (KOM) i może być jedną z zaprezentowanych wartości porozdzielanych znakiem '/',

id, **topic**, **mode**: zawierają informacje identyfikujące odpowiednio usługę i temat komunikacji oraz tryb producenta lub subskrybenta,

timestamp: zawiera znacznik czasowy uzyskania danych będących przedmiotem transmisji (lub chwili wygenerowania komunikatu) zapisany jest w zgodnie ze standardem ISO 8601 i jednoznacznie identyfikuję komunikat (KOM),

payload: zawiera treść komunikatu (KOM) i format jest różny w zależności od typu komunikatu, co będzie definiowane poniżej w przypadku opisywania poszczególnych funkcji,

następnie

- gdy walidacja KOM jest negatywna, umieszcza w KKW obiekt z przypisanym gniazdem klienta (nadawcy wadliwego komunikatu) i wiadomością o typie „**reject**" w temacie „**logs**":

```
{
  "type": "reject",
  "id": "ServerID",
  "topic": "logs",
  "timestamp": "2024-02-15T00:00:01.500Z",
  "payload": {
    "timestampOfMessage": "2024-02-15T00:00:01.000Z",
    "topicOfMessage": "lunch call",
    "success": false,
    "message": "The message was unpleasant"
  }
}
```

gdzie w polach **timestampOfMessage** oraz **topicOfMessage** należy umieścić skopiowane z komunikatu (KOM) klienta wartości pól odpowiednio **timestamp** oraz **topic**,

Możliwe błędy walidacji formatu:

- ✓ błąd parsowania JSON komunikatu, co dodatkowo może wynikać np. ze zbyt dużego rozmiaru komunikatu, co w tym przypadku powoduje, że komunikat został podzielony i przestał być zgodny z formatem JSON,
- ✓ brak któregoś z pól lub nieznany typ danych np. **id** jest typu **int**,
- ✓ nieznany typ wiadomości,
- ✓ spodziewany **mode=producer** przy rejestracji nowego lub wycofywaniu tematu,

- ✓ błędny format wartości znacznika czasowego,
- ✓ gdy typ **aknowledge** lub **reject**, to w payload obowiązkowe pola **timestampOfMessage, topicOfMessage, success, message,**
- ✓ itp.

- gdy walidacja KOM jest pozytywna, to uruchamia funkcję zarządzania komunikatem,

- zarządzanie komunikatem KOM (uwarunkowane pozytywnym wynikiem walidacji) i jeśli typ KOM jest:
  - **register**,
    to w zależności od pozostałych informacji KOM może być rozważana następująca sytuacja:
    - ✓ rejestracja nowego tematu w LT, gdzie dodatkowo należy zapamiętać referencję do gniazda klienta-producenta oraz jego ID,
    - ✓ odrzucenie żądania rejestracji tematu, gdy taki temat już istnieje i został utworzony przez innego producenta,
    - ✓ rejestracja subskrybenta tematu, gdzie do istniejącego tematu w LT należy dodać referencję do gniazda nowego klienta-subskrybenta,
    - ✓ odrzucenie żądania subskrypcji, jeśli temat nie istnieje,

    a następnie:
    - umieszczenie w KKW obiektu z przypisanym gniazdem klienta i komunikatem z potwierdzeniem typu **acknowledge** na temat **logs** z następującą informacją w polu payload:

```
{
    "timestampOfMessage": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": true,
    "message": "OK"
}
```

    lub typu **reject** na temat **logs** z następującą informacją w polu payload:

```
{
    "timestampOfMessage ": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": false,
    "message": "no such topic registered"
}
```

UWAGA: temat **logs** jest specjalnym tematem, który powinien być automatycznie zarejestrowany w LT jako pierwszy zaraz po uruchomieniu wątku zarządzania komunikatem i do tego tematu jako producenta należy zarejestrować usługę aplikacji SP z pustą referencją do gniazda (serwer wysyła komunikaty na **logs** i umieszczając w polu **mode** wartość '**producent**', a w polu **payload.message** dokładną przyczyną błędu lub informację o powodzeniu),
UWAGA: każdy podłączony klient może być producentem wielu tematów i podobnie subskrybować wiele tematów, dodatkowo ta cecha powinna dopuszczać możliwość jednoczesnej subskrypcji własnego tematu,

  - **withdraw**
    to w zależności od pozostałych informacji KOM może być rozważana następująca sytuacja:
    - ✓ wycofanie tematu, jeśli nadawcą KOM jest jego producent i to oznacza usunięcie wpisu tematu z LT (patrz dodatkowo informację zawartą w ramce: ROZŁĄCZANIE KLIENTÓW)

    oraz umieszczenie w KKW obiektu z przypisaną tablicą zawierającą referencje do gniazd producenta i wszystkich subskrybentów podłączonych do tematu i z komunikatem typu **acknowledge** na temat **logs** z następującą informacją w polu **payload**:

```
{
    "timestampOfMessage": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": true,
    "message": "Producer withdrawed the topic"
}
```

    lub obiektu z przypisanym gniazdem nadawcy i z komunikatem typu **reject** na temat **logs** z następującą informacją w polu **payload**:

```
{
    "timestampOfMessage ": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": false,
    "message": "no topic registered"
}
```

✓ wycofanie subskrypcji tematu, jeśli nadawcą KOM jest jego subskrybent i to oznacza usunięcie wpisu subskrypcji tematu w LT
oraz umieszczenie w KKW obiektu z przypisanym gniazdem klienta i z komunikatem typu **acknowledge** na temat **logs** z następującą informacją w polu **payload**:

```
{
    "timestampOfMessage": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": true,
    "message": "Subscription of the topic withdrawed"
}
```

lub obiektu z przypisanym gniazdem nadawcy i z komunikatem typu **reject** na temat **logs** z następującą informacją w polu payload:

```
{
    "timestampOfMessage ": "2024-02-15T00:00:01.000Z ",
    "topicOfMessage": "lunch call",
    "success": false,
    "message": "no topic registered"
}
```

ROZŁĄCZANIE KLIENTÓW:
Należy zwrócić szczególną uwagę na potrzebę implementacji działań związanych z rozłączaniem połączeń z klientem w kontekście zarejestrowanych tematów i powinno to odbywać się w dwóch aspektach:

1. gdy rozłączenie połączenia nastąpiło z inicjatywy klienta i wówczas należy przeszukać LT i usunąć rejestrację klienta ze wszystkich subskrybowanych tematów oraz jeżeli klient był producentem tematu, to należy usunąć ten temat z LT,
2. usunięcie tematu z LT powinno skutkować rozłączeniem aktywnych połączeń z klientami, którzy ten temat subskrybowali ale tylko w sytuacji, gdy dany klient równolegle nie subskrybuje innego tematu lub nie jest innym producentem,

- **message**, **file**
  to należy umieścić w KKW obiekt z przypisaną tablicą zawierającą referencje do gniazd wszystkich subskrybentów podłączonych do tematu i z komunikatem, który jest wierną kopią KOM,
  oraz dodatkowo umieścić w KKW obiekt z przypisanym gniazdem klienta (producenta) i z komunikatem typu **acknowledge** na temat **logs** z następującą informacją w polu payload:

```
{
    "timestampOfMessage": "2024-02-15T00:00:02.000Z ",
    "topicOfMessage": "lunch call",
    "success": true,
    "message": "Message resent to 3 subscribers"
}
```

  lub innym komunikatem typu **reject** z polem **success** równym **false**, jeśli wystąpił problem np. temat nie został zarejestrowany, lub brak subskrybentów itp. i wówczas w polu **message** należy opisać problem i ewentualnie przyczynę,
- **config**
  to w KKW należy umieścić obiekt z przypisanym gniazdem klienta (nadawcy) i komunikatem typu **config** na temat **logs** załączając w polu **payload** zawartość pliku `config.json` i jest to potrzebne w kontekście pilnowania przez klienta, aby nie przekroczyć dozwolonego rozmiaru jednego komunikatu,
- **status**
  to w KKW należy umieścić obiekt z przypisanym gniazdem klienta (nadawcy) i komunikatem typu **status** na temat **logs** załączając w polu **payload** informacje o obecnie zarejestrowanych tematach wraz z ID ich producentów,

  - usuwanie przetworzonego KOM z KKO,

- wątek interfejsu komunikacji z użytkownikiem:
  - nasłuchuje/odczytuje oraz interpretuje polecenia/komendy przekazywane przez użytkownika,
  - interfejs użytkownika może być w postaci GUI lub konsoli,
  - interfejs użytkownika prezentowany w języku angielskim powinien umożliwiać:
    - zamykanie wątku serwera wraz z zamknięciem wszystkich pracujących wątków klientów i rozłączeniem ich poprzez ustawienie flag STOP_SERVER oraz STOP_CLIENT,
    - wypisywać informacje o numerze portu TCP i interfejsach sieciowych, na którym realizowane jest nasłuchiwanie serwera SP,
    - wypisywać szczegółowe informacje o wszystkich zarejestrowanych tematach wraz z ID producentów i subskrybentów,

API Klienta SP powinno udostępniać następujące metody z parametrami:

- `Start (serverIP, serverPort, clientID):`
  - jako parametr wywołania dostarcza użytkownikowi możliwości ręcznego wprowadzenia punktu końcowego (tj. adresu IP i numeru portu TCP) Serwera SP,
  - zapamiętuję swoją nazwę **clientID** jako wartość do wstawiania w pole **id** wysyłanego komunikatu,
  - nawiązuje połączenie z serwerem SP,
  - uruchamia wątek W1, w którym realizuje nasłuchiwanie przysyłanych od serwera komunikatów,
  - po poprawnym nawiązaniu połączenia z serwerem, wysyła do serwera komunikat typu **config** w temacie **logs** i po otrzymaniu konfiguracji zapamiętuję ustawienia serwera,

FORMAT WYSYŁANYCH KOMUNIKATÓW:
Należy zwrócić szczególną uwagę, aby w ramach implementacji kolejnych opisywanych funkcji, które realizują wysyłanie danych do serwera SP w komunikatach były poprawnie uzupełniane pola: **id**, **topic**, **mode** oraz **timestamp**,

- `bool IsConnected ()`
  - zwraca bieżący status połączenia z serwerem,
- `string GetStatus ()`
  - zwraca w formacie JSON informację o własnych produkowanych tematach i aktywnych subskrypcjach,
- `GetServerStatus (callback)`
  - wysyła do serwera komunikat typu **status** w temacie **logs** i po otrzymaniu wiadomości uruchamia zarejestrowaną funkcją (**callback**) np. wypisującą na ekranie wartości pola **payload** zawierającego listę tematów zarejestrowanych po stronie serwera,
- `GetServerLogs (callback)`
  - porównuje wartości pół **timestamp** wysłanych komunikatów z polem **timestampOfMessage** komunikatów powrotnych i analizuje, czy wysłane komunikaty zostały potwierdzone czy odrzucone,
  - rejestruje funkcję (callback) (np. wypisującą na ekranie otrzymane jako parametr informacje) uruchamianą w chwili otrzymania komunikatu typu **acknowledge** i **reject**, której jako parametr przekazuje wynik analizy z poprzedniego punktu (success true lub false) oraz dodatkowo wartość pola **message** obiektu **payload**,
- `CreateProducer (topicName)`
  - wysyła do serwera komunikat typu **register** z prośbą o rejestrację producenta tematu **topicName**,
- `Produce (topicName, payload)`
  - wysyła do serwera komunikat typu **message** na temat **topicName** umieszczając w polu **payload** kopie parametru **payload**,
- `SendFile (topicName, path2File)`
  - czyta plik (w trybie binarnym) z podanej lokalizacji **path2File** i w polu **payload** umieszcza jego zawartość wraz z informacją o nazwie pliku, a następnie wysyła do serwera komunikat typu **file** na temat **topicName** przy czym zwraca uwagę, aby rozmiar komunikatu nie przekroczył dopuszczalnej wartość otrzymanej od serwera (**SizeLimit**),
- `WithdrawProducer (topicName)`
  - wysyła do serwera komunikat typu **withraw** z prośbą o wycofanie produkowania w temacie **topicName**,
- `CreateSubscriber (topicName, callback)`
  - wysyła do serwera komunikat typu **register** z prośbą o rejestrację subskrypcji tematu **topicName**,
  - rejestruje funkcję (**callback**) uruchamianą w chwili otrzymania komunikatów na subskrybowany temat **topicName** (np. do wypisania na ekranie zawartości komunikatu) przekazując do niej jako parametr zawartość pola **payload** komunikatu,
  - prowadzi rejestr subskrybowanych tematów z przypisaniem **callback**,
  - jeżeli typ komunikatu jest **file**, to dodatkowo w funkcji **callback** należy zaimplementować tworzenie nowego pliku pod nazwą zapisaną w komunikacie i z otrzymaną zawartością,
- `WithdrawSubscriber (topicName)`
  - wysyła do serwera komunikat typu **withraw** z prośbą o wycofanie subskrybcji tematu **topicName**,
  - usuwa zarejestrowany **callback** związany z otrzymywaniem komunikatów w temacie **topicName**,
- `Stop`
  - kończy działanie wątku W1, rozłącza połączenie z serwerem SP, wyrejestrowuje wszystkie callback'i.

Należy dodatkowo zaimplementować przykładowy program importujący API oraz demonstrujący uruchamianie poszczególnych metod API.

Program demonstracyjny implementujący API Klienta SP:

- powinien zaprezentować wywołania wszystkich udostępnionych metod API oraz dodatkowo,
- powinien zostać producentem i jednocześnie subskrybentem przynajmniej jednego tematu,
- ale mógłby być też testowany, jako uruchomione dwie instancje i wówczas na przemian produkować i subskrybować dwa różne tematy.

Plik `config.json` - przykład

```
{
  "ServerID": "it's me, your server",
  "ListenAddresses": ["*"],
  " ListenPort": 1234,
  "TimeOut": 3,
  "SizeLimit": 1048576,
  " AllowedIPAdresses": ["10.0.0.0/24", "192.168.1.0/24"]
}
```

Dostępne wartości pola ListenAddresses:
["*"] – oznacza możliwość powiązania do gniazda wszystkich dostępnych interfejsów,
["localhost", "127.0.0.1", "192.168.1.2"] – oznacza konieczność powiązania do gniazda tylko wyspecyfikowanych interfejsów.
Pole AllowedIPAdresses to tablica adresów IP zapisanych w notacji CIDR do weryfikacji, czy połączenia klientów pochodzą z uprawnionych sieci komputerowych i jeśli wartość any, to nie jest realizowana weryfikacja IP.

Wszystkie moduły (tj. serwer SP, API Klienta oraz Program demonstracyjny) muszą być odporne na niewłaściwe dane wpisywane przez użytkownika.

Uwagi do nadsyłanych programów

1. Program powinien spełniać wszystkie kryteria opisane powyżej, w tym:
   - kolorem czarnym obowiązkowe na ocenę 3,
   - kolorem bordowym na ocenę 3,5
     (tj. funkcjonalności związane z konfiguracją pracy serwera i zamykaniem jego wątków),
   - kolorem niebieskim na 4
     (tj. funkcjonalności związane z logami),
   - kolorem zielonym na 5
     (tj. funkcjonalności związane z transmisją plików oraz filtrowaniem dozwolonych adresów IP),

   funkcjonalności określone na oceny 4 i 5 mogą być zaimplementowane zamiennie,
2. Zaimplementowana procedura obsługi błędów i sytuacji wyjątkowych,
3. Czytelny i przejrzysty kod z utrzymaną notacją nazw zmiennych, metod oraz komentarzy w języku angielskim.

Dodatkowe informacje

- ObservableCollection w C#: https://learn.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1.collectionchanged?view=net-8.0&redirectedfrom=MSDN
- notacja CIDR dla adresów IPv4,
- sprawdzenie, czy adres należy do podsieci:
  - C: https://stackoverflow.com/questions/31040208/standard-safe-way-to-check-if-ip-address-is-in-range-subnet
  - Py: https://netaddr.readthedocs.io/en/latest/tutorial_01.html
  - Java: https://stackoverflow.com/questions/577363/how-to-check-if-an-ip-address-is-from-a-particular-network-netmask-in-java
  - https://xion.org.pl/2012/02/04/checking-whether-ip-is-within-a-subnet/