Programowanie Aplikacji Sieciowych - Laboratorium 15

Serwery zdarzeniowe (event-based lub event-driven servers):

- Istotną wadą programowania z użyciem wątków jest problem pojawiający się w sytuacji, gdy istnieje potrzeba obsługi bardzo dużej liczby połączeń - dla każdego nowego połączenia, które należy obsłużyć, należy utworzyć nowy wątek. Każdy nowy wątek zużywa zasoby (pamięć, CPU ...) oraz dodaje narzut związany z przełączaniem między wątkami (tzw. *context switching*) i ich synchronizacją.
- Programowanie wielowątkowe sprawdza się w sytuacji, gdy mamy do obsługi stosunkowo niewielką liczbę wątków, jednakże duża liczba wątków do obsłużenia wpływa znacząco na wydajność. Multiprocesssing charakteryzuje się podobnymi problemami.
- Alternatywą dla multithreading i multiprocessing jest wykorzystanie modelu opartego na zdarzeniach. W tym modelu, zamiast polegać na systemie operacyjnym, którego zadaniem jest przełączanie pomiędzy aktywnymi wątkami czy procesami, wykorzystujemy pojedynczy wątek, rejestrując w nim obiekty blokujące - takie jak na przykład sockety.
- Nasz program jest cały czas „bombardowany" zdarzeniami (events), na które musi odpowiedzieć - przepływ sterowania w programie jest całkowicie niemożliwy do przewidzenia z góry.
- W programowaniu opartym na zdarzeniach mamy jeden wątek przetwarzania, który po prostu oczekuje i reaguje na wszelkiego typu zdarzenia (ktoś się połączył, odebrano jakieś dane, coś można już wysyłać, ...). Uporawszy się z obsługą jednego zdarzenia, program sprawdza, czy nadeszło następne. I tak dalej. Nie ma żadnej synchronizacji czy rywalizacji o zasoby, bo w każdym momencie działa tylko jedna funkcja (i nic jej nie przerywa).

Serwer zdarzeniowy w Pythonie:

- **funkcja** `select` (działa na Win i Linux) - starsza i wolniejsza, funkcja `select` jest bezpośrednim interfejsem do implementacji dostępnej w systemie operacyjnym, odpowiada za monitorowanie gniazd, otwartych plików oraz strumieni, dopóki nie staną się gotowe do zapisu lub odczytu, lub, dopóki nie wystąpi błąd, funkcja `select` pozwala na monitorowanie wielu połączeń sieciowych jednocześnie, w Pythonie moduł `select` zawiera funkcję `select`, która umożliwia aplikacji na oczekiwanie danych z różnych gniazd w tym samym czasie.

  Argumenty funkcji `select`:

  ```
  in, out, err = select(input, output, exception [,timeout])
  ```

  Pierwsze trzy argumenty to listy gniazd lub obiektów plików, które oczekują na dane wejściowe (`input`) lub wyjściowe (`output`) lub wyjątek (`exception`). Jeżeli nie zostanie podany `timeout` - maksymalny czas działania, to wywołanie `select` będzie zablokowane dopóki na jednym z listowanych gniazd nie wydarzy się wyjątek/dane wejściowe/wyjściowe. Wartość zero oznacza, że wywołanie nie będzie zablokowane jeżeli żadne z gniazd jest gotowe.

  Metoda `select` zwraca tuplę składającą się z trzech list: listę gniazd i plików, na których zaszły zdarzenia. Jeżeli został przekroczony maksymalny czas bez zajścia zdarzeń wszystkie listy będą puste.

- **funkcja** `poll` (działa na Linux) - prosta w obsłudze,
- **funkcja** `epoll` (działa na Linux) - wydajniejsza, ale bardziej skomplikowana

Przykład serwera wykorzystującego funkcję `select`:

```
import select, socket, Queue

if __name__ == "__main__":

    # tworzymy gniazdo serwera
    # setblocking -> False
    # bind, listen

    inputs = [server_socket]
    outputs = []

    message_queues = {}

    while inputs:

        readable, writable, exceptional = select.select(inputs, outputs, inputs)

        # jesli zdarzenie nastapilo w liscie do czytania
        for s in readable:

            # jesli zdarzenie wywolalo gniazdo serwera
            if s is server_socket:

                # akceptowanie polaczenia klienta, setblocking -> False
                # dodajemy klienta do listy inputs
                # dodajemy do kolejki wiadomosci dla danego klienta nowa kolejke

            # jesli zdarzenie wywolalo gniazdo klienta
            else:

                # odbieramy dane za pomoc recv, jesli przyszlo cos nie pustego, to ...
                if data:

                        # wrzucamy na kolejke wiadomosci danego klienta

                    # jesli gniazda klienta nie ma w liscie do pisania ....
                    if s not in outputs:
                        # ... dodajemy

                # jesli nic nie przyszlo -> ktos sie rozlaczyl
                else:

                        # jesli giazdo bylo w liscie do pisania ....
                        if s in outputs:
                               # ... usuwamy je

                        # usuwamy gniazdo z listy do czytania
                        # zamykamy gniazdo
                        # usuwamy kolejke wiadomosci gniazda

        # jesli zdarzenie nastapilo w liscie do pisania
        for s in writable:
            try:
                # pobierz z kolejki wiadomosc
            except Queue.Empty:
                # w przypadku bledu, usun gniazdo z listy do pisania
            else:
                # wyslij wiadomosc uzywajac sendall

        # jesli zdarzenie nastapilo w liscie odpowiedzialnej za monitorowanie bledow
        for s in exceptional:
            inputs.remove(s)
            if s in outputs:
                outputs.remove(s)
            s.close()
            del message_queues[s]
```

Komentarz:

- Kod blokującego, zdarzeniowego serwera jest nieco dłuższy niż kod serwera wykorzystującego wątki
- Wynika to z faktu, iż w programie musimy utrzymywać i monitorować 3 różne listy gniazd - gniazda czekające na dane wejściowe, wyjściowe, oraz listę gniazd, podczas działania których zaobserwowano błąd.
- Utworzenie nasłuchującego gniazda serwera (`server_socket`) wygląda identycznie, jak do tej pory, różnica jest właściwie tylko jedna - należy przestawić gniazdo serwera w tryb blokujący, wywołując metodę `server_socket.setblocking(0)`
- Wywołując funkcję `select`, pytamy system operacyjny, czy gniazda gotowe są do odbierania / wysyłania wiadomości (lub czy zaobserwowano błąd), w odpowiedzi dostajemy listy, które OS będzie monitorował, aby sprawdzić, czy nie nastąpiło zdarzenie (pisania / czytania / błędu)
- Kolejnym krokiem programu jest sekwencyjne iterowanie po listach i wykonywanie odpowiednich akcji. Przykładowo, gdy zdarzenie czytania zostanie zaobserwowane w gnieździe serwera, będzie to oznaczać, że z serwerem połączył się nowy klient (linia `24`) - możemy wówczas wywołać metodę `accept`, dodać gniazdo zaakceptowanego klienta go listy gniazd czekających na dane wejściowe (linia `25`) i tworzymy dla klienta kolejkę na przychodzące wiadomości (linia `26`)
- Jeśli zdarzenie czytania wywołało gniazdo klienta, wówczas oznacza to, że któryś z klientów przysłał dane do serwera - można je odebrać i umieścić w odpowiedniej kolejce
- Dla zdarzeń związanych z wysyłaniem wiadomości, jeśli socket jest w stanie writable, oznacza to, że można pobrać z kolejki wiadomość (jeśli takowa się w niej znajduje), a następnie ją przesłać
- Tak działa to na najniższej warstwie. Oczywiście, dostępne są gotowe biblioteki, które ułatwiają to zadanie, np. `Tornado`, `Twisted` czy `ZeroMQ`

Uwaga W poniższych zadaniach zakładamy, że jeśli w zadaniu trzeba napisać serwer, powinien on obsługiwać kilku klientów jednocześnie, jednak bez użycia wątków.

1. Napisz program serwera, który działajac pod adresem `127.0.0.1` oraz na określonym porcie TCP, dla podłaczającego się klienta, będzie odsyłał mu przesłaną wiadomość (tzw. serwer echa). Serwer powinien móc obsługiwać kilku klientów jednocześnie (bez użycia wątków).

   Napisz program klienta, który połączy sie z serwerem TCP działającym pod adresem `127.0.0.1` na określonym porcie TCP, a następnie będzie w pętli wysyłał do niego tekst wczytany od użytkownika, i odbierał odpowiedzi.

2. Napisz program serwera, który działajac pod adresem `127.0.0.1` oraz na określonym porcie TCP, dla podłaczającego się klienta, będzie odsyłał mu informacje o aktualnej pogodzie w Lublinie. Serwer powinien móc obsługiwać kilku klientów jednocześnie (bez użycia wątków).

   Serwer do pobrania danych dotyczących pogody, może wysyłać odpowiednio sformatowane requesty HTTP do serwera `openweathermap.org`, który udostępnia API pod adresem https://openweathermap.org/current. Nie musisz rejestrować nowego konta, możesz wykorzystać następujący klucz API: `d4af3e33095b8c43f1a6815954face64`. (Klucz został wygenerowany dla adresu e-mail: `pasinf2017@gmail.com`, nazwa użytkownika to `PAS2017`, hasło `P4SInf2017`.)

   Nie wykorzystuj żadnych dodatkowych bibliotek (jedyne dozwolone to te wykorzystujące gniazda, i ewentualnie odpowiedzialne za parsowanie formatu `xml` i/lub `json`).

   Dopisz program klienta, który łącząc się z serwerem będzie wysyłał do niego zapytanie o pogodę.

   Zaimplementuj własny protokół (dozwolone komendy, zakończenie wiadomości, ...).