Programowanie sieciowe

Miłosz Cieśla, Filip Ryniewicz, Aleksander Szymczyk

Sprawozdanie z zadania 2

29.11.2024

1 Treść zadania

Z 2 - Komunikacja TCP

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Transmitowany strumień danych powinien być stosunkowo duży, nie mniej niż 100 kB.

Zmodyfikować serwer tak, aby miał konstrukcję współbieżną, tj. obsługiwał każdego klienta w osobnym procesie. Przy czym:

- Dla C należy posłużyć się funkcjami fork() oraz (obowiązkowo) wait().
- Dla Pythona należy posłużyć się wątkami, do wyboru: wariant podstawowy lub skorzystanie z ThreadPoolExecutor.

Przetestować dla kliku równolegle działających klientów.

2 Opis rozwiązania

W programie w języku C używana jest obsługa współbieżności przy pomocy funkcji `fork()` oraz `wait()`.
W programie w języku Python używana jest obsługa współbieżności przy pomocy puli wątków z `ThreadPoolExecutor`, z maksymalną liczbą jednocześnie działających wątków ustawioną na 5.

2.1 Schemat działania

- Tworzonych jest 10 instancji klientów.
- Serwer i klienci tworzą własne sockety.
- Serwer podpina swój socket pod wybrany przez argument wywołania port.
- Serwer przygotowuje socket do nasłuchu ustalając długość kolejki połączeń oczekujących na 5.
- Serwer oczekuje na zaakceptowanie nadchodzących połączeń w pętli.
- Każdy klient generuje wiadomość o rozmiarze `100kB` składającą się z powtarzającego się alfabetu.

2.1.1 Python

- Kiedy klient się połączy:
  - Tworzony jest nowy wątek dla obsługi tego klienta.
  - Wątek odbiera dane od klienta i zapisuje je do bufora.
  - Odsyła tą samą wiadomość w odpowiedzi do klienta.
  - Po zakończeniu zamyka połączenie z klientem.
- Główna pętla serwera pozostaje wolna do obsługi kolejnych połączeń.

2.1.2 C

- Kiedy klient się połączy:
  - Serwer wywołuje fork()
  - Proces potomny:
    - Obsługuje połączenie klienta.
    - Odbiera dane, zapisuje wiadomość do bufora.
    - Odsyła tą samą wiadomość w odpowiedzi do klienta.
    - Zamyka socket i kończy działanie.
  - Proces macierzysty:

    - Zamyka socket klienta (nie obsługuje go).
    - Wraca do oczekiwania na kolejne połączenia.

- Serwer obsługuje wielu klientów jednocześnie dzięki oddzielnym procesom. Procesy potomne są czyszczone.

Przyjeliśmy maksymalną długość kolejki dla funkcji accept() jako 5.
Struktura projektu została opisana w README.md. Kod zawiera komentarze opisujące jego działanie.

3 Problemy

- Pierwszy klient nie łączył się z serwerem, rozwiązaliśmy to przez dodanie parametru wywołania odpowiedzialnego za opóźnienie wykonania klienta. Obecnie uruchamiamy wszystkich klientów 1 sekundę po serwerze.
- Wszystkie wywołania były obsługiwane na jednym wątku, co zauważyliśmy, ponieważ `thread_id` miały taką samą wartość. Problem wynikał z tego, że operacje connect/accept zajmowały dłużej niż to co powinno się wykonywać na wątkach. Rozwiązaliśmy to poprzez dodanie opóźnienia 0.5s do operacji `handle_client()`, pomaga to zobrazować działanie współbieżności w celach demonstracyjnych.
- Odpowiedź do klienta była przesyłana fragmentami, przez co rozmiar każdego z nich wyświetlany był na terminalu. Prowadziło to do nieczytelności testów, więc zmieniliśmy funkcję tak, by rozmiar wszystkich fragmentów odpowiedzi był sumowany. Po zebraniu wszystkich fragmentów odpowiedzi na terminalu wyświetlony jest jej cały rozmiar.

4 Konfiguracja testowa

Rozwiązanie było testowane w dwóch wariantach:

4.1 Lokalnie

Utworzono sieć docker typu bridge o nazwie `z40_network`. Serwer przyjmował adres IP: `172.20.0.2`, a klienci: kolejne adresy rozpoczynając od `172.20.0.3`. Serwer TCP uruchamiany był na porcie `8000`, który to port był następnie podawany do klienta wraz z nazwą kontenera realizującego serwer.

4.2 Na serwerze `bigubu.ii.pw.edu.pl`

Do testów wykorzystywano utworzoną wcześniej sieć o nazwie `z40_network`. Serwer przyjmował adres IP: `172.21.40.4`, a klienci: kolejne adresy rozpoczynając od `172.21.40.5`. Serwer TCP uruchamiany był na porcie `8000`, który to port był następnie podawany do klienta wraz z nazwą kontenera realizującego serwer.

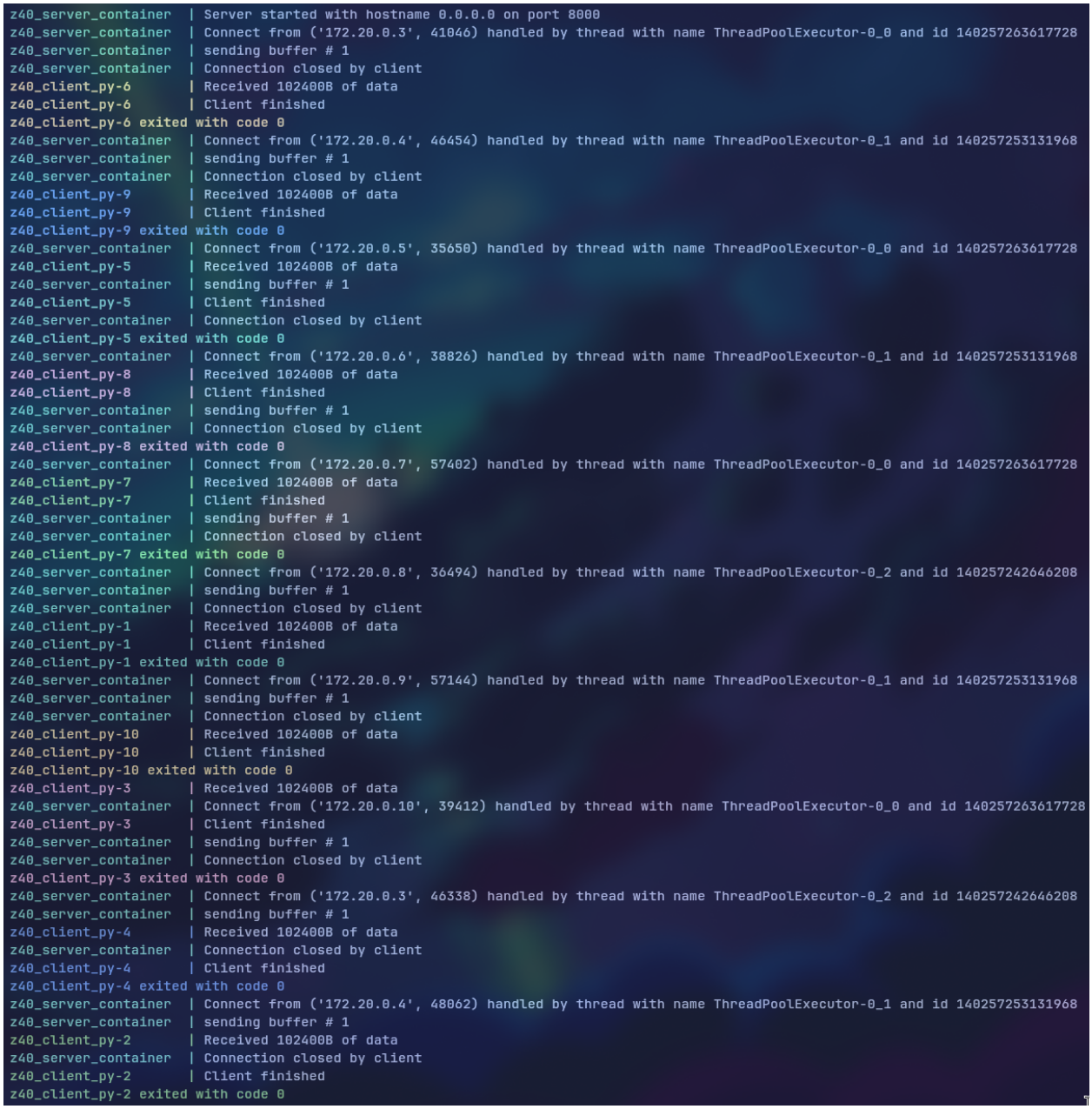
5 Testy

Połączenie TCP było testowane przez utworzenie wielu klientów, którzy równolegle nawiązywali połączenia z serwerem. Po poprawnym zestawieniu połączenia, na terminalu wyświetlany był adres IP oraz port klienta, a także ID wątku (w przypadku implementacji w Pythonie) lub ID procesu (w przypadku implementacji w C), które odpowiadały za obsługę konkretnego klienta. Następnie te same dane są odsyłane z powrotem do klienta, który wyświetla liczbę bajtów otrzymanej odpowiedzi na terminal (poprawna gdy odpowiedź ma rozmiar `100kb`).

5.1 Testy lokalne

Przyjęty został delay 1 sekundu przed uruchomieniem klienta.

5.1.1 Python



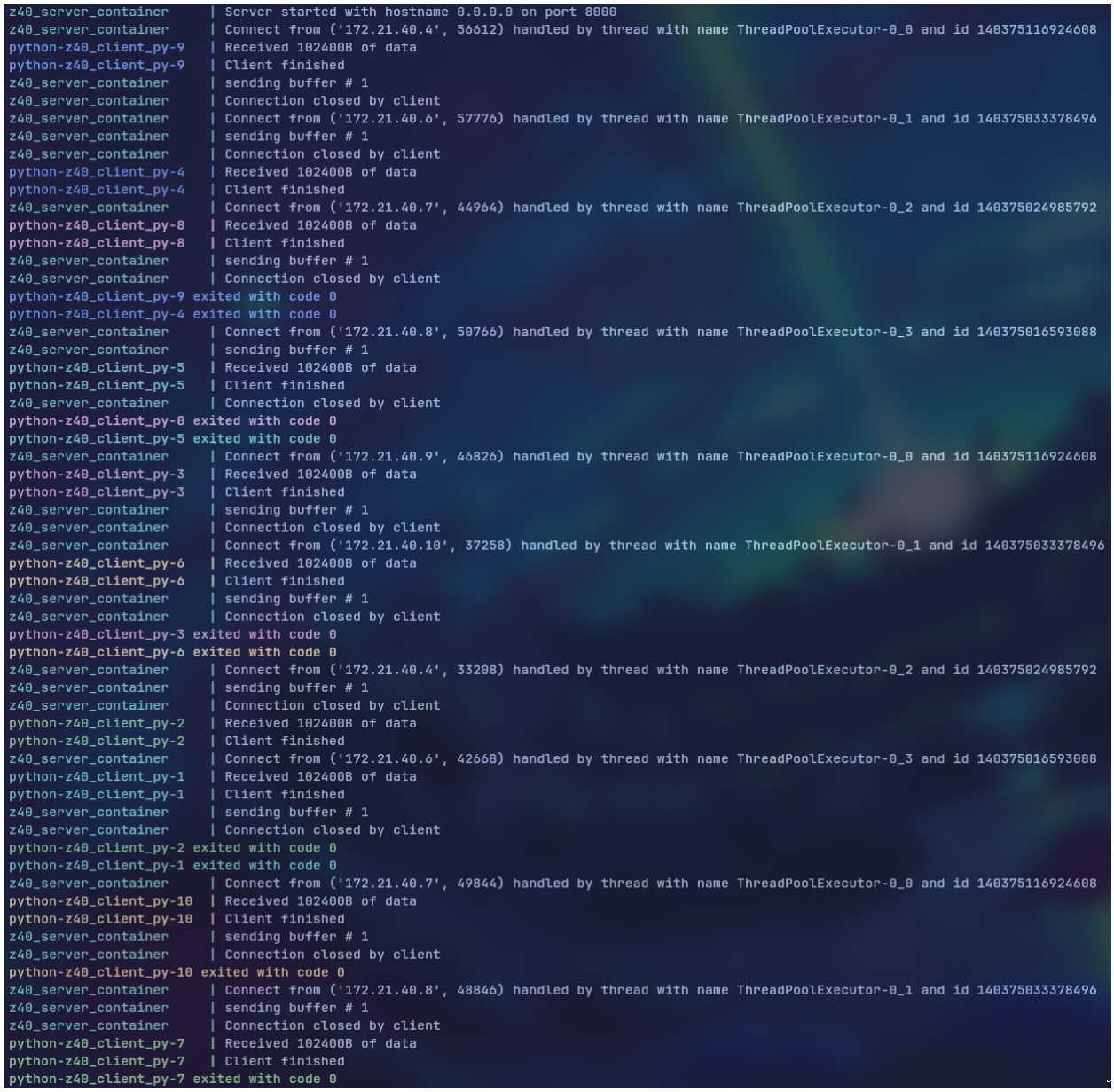
5.1.2 C

```
z40_server_c_container  | Server started and listening on port #8000
z40_server_c_container  | Connected to client 172.20.0.3:39972
z40_server_c_container  | Handling client communication on PID: 7
z40_client_c-9          | Received 102400 bytes from server
z40_server_c_container  | Connected to client 172.20.0.4:46356
z40_server_c_container  | Handling client communication on PID: 8
z40_client_c-9 exited with code 0
z40_server_c_container  | Connected to client 172.20.0.5:47296
z40_server_c_container  | Handling client communication on PID: 9
z40_client_c-4          | Received 102400 bytes from server
z40_client_c-4 exited with code 0
z40_server_c_container  | Connected to client 172.20.0.6:51464
z40_server_c_container  | Handling client communication on PID: 10
z40_client_c-6          | Received 102400 bytes from server
z40_client_c-6 exited with code 0
z40_client_c-1          | Received 102400 bytes from server
z40_server_c_container  | Connected to client 172.20.0.7:53460
z40_server_c_container  | Handling client communication on PID: 11
z40_client_c-1 exited with code 0
z40_server_c_container  | Connected to client 172.20.0.8:59862
z40_client_c-3          | Received 102400 bytes from server
z40_server_c_container  | Handling client communication on PID: 12
z40_client_c-3 exited with code 0
z40_server_c_container  | Connected to client 172.20.0.9:56666
z40_server_c_container  | Handling client communication on PID: 13
z40_client_c-7          | Received 102400 bytes from server
z40_client_c-7 exited with code 0
z40_client_c-5          | Received 102400 bytes from server
z40_server_c_container  | Connected to client 172.20.0.10:37802
z40_server_c_container  | Handling client communication on PID: 14
z40_client_c-5 exited with code 0
z40_client_c-8          | Received 102400 bytes from server
z40_server_c_container  | Connected to client 172.20.0.11:51600
z40_server_c_container  | Handling client communication on PID: 15
z40_client_c-8 exited with code 0
z40_server_c_container  | Connected to client 172.20.0.12:58634
z40_server_c_container  | Handling client communication on PID: 16
z40_client_c-2          | Received 102400 bytes from server
z40_client_c-2 exited with code 0
z40_client_c-10         | Received 102400 bytes from server
z40_client_c-10 exited with code 0
```

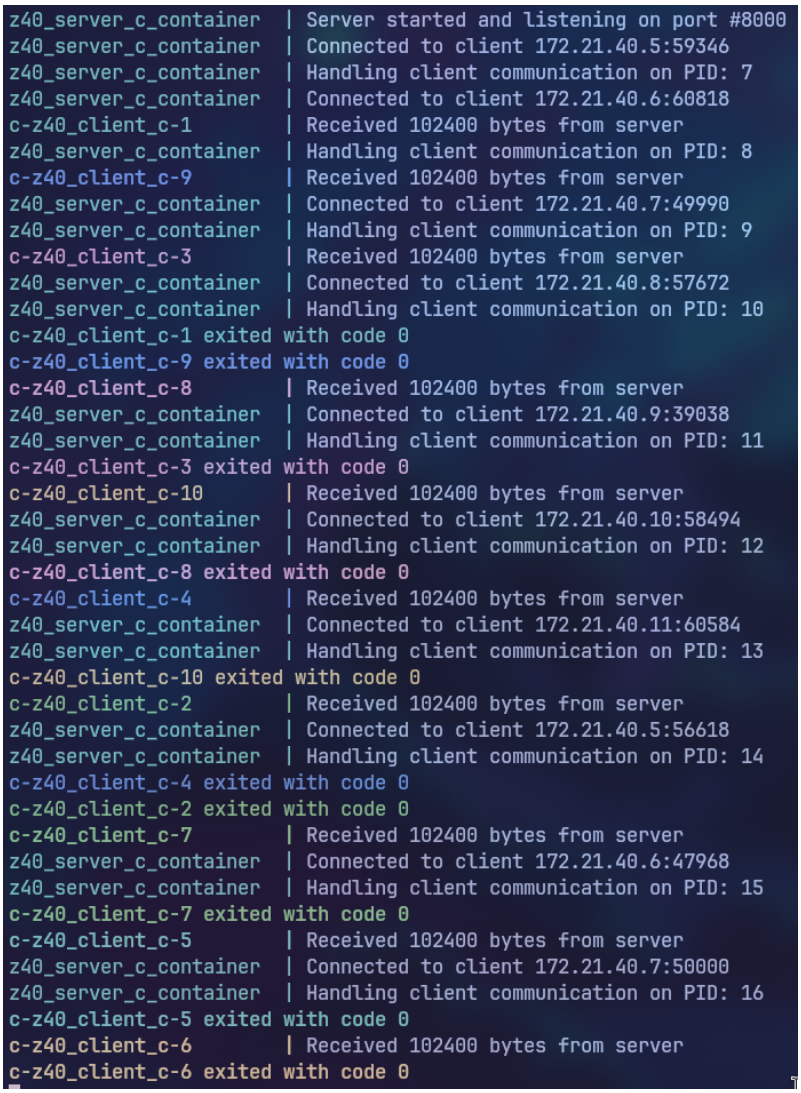
5.2 Testy na serwerze `bigubu.ii.pw.edu.pl`

Przyjęty został delay 5 sekund przed uruchomieniem klienta, różnica względem wartości dla testów lokalnych wynika z tego, że bigubu jest wolniejsze.

5.2.1 Python


z40_server_container | Server started with hostname 0.0.0.0 on port 8000
z40_server_container | Connect from ('172.21.40.4', 56612) handled by thread with name ThreadPoolExecutor-0_0 and id 140375116924608
python-z40_client_py-9 | Received 102400B of data
python-z40_client_py-9 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
z40_server_container | Connect from ('172.21.40.6', 57776) handled by thread with name ThreadPoolExecutor-0_1 and id 140375033378496
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-4 | Received 102400B of data
python-z40_client_py-4 | Client finished
z40_server_container | Connect from ('172.21.40.7', 44964) handled by thread with name ThreadPoolExecutor-0_2 and id 140375024985792
python-z40_client_py-8 | Received 102400B of data
python-z40_client_py-8 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-9 exited with code 0
python-z40_client_py-4 exited with code 0
z40_server_container | Connect from ('172.21.40.8', 50766) handled by thread with name ThreadPoolExecutor-0_3 and id 140375016593088
z40_server_container | sending buffer # 1
python-z40_client_py-5 | Received 102400B of data
python-z40_client_py-5 | Client finished
z40_server_container | Connection closed by client
python-z40_client_py-8 exited with code 0
python-z40_client_py-5 exited with code 0
z40_server_container | Connect from ('172.21.40.9', 46826) handled by thread with name ThreadPoolExecutor-0_0 and id 140375116924608
python-z40_client_py-3 | Received 102400B of data
python-z40_client_py-3 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
z40_server_container | Connect from ('172.21.40.10', 37258) handled by thread with name ThreadPoolExecutor-0_1 and id 140375033378496
python-z40_client_py-6 | Received 102400B of data
python-z40_client_py-6 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-3 exited with code 0
python-z40_client_py-6 exited with code 0
z40_server_container | Connect from ('172.21.40.4', 33208) handled by thread with name ThreadPoolExecutor-0_2 and id 140375024985792
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-2 | Received 102400B of data
python-z40_client_py-2 | Client finished
z40_server_container | Connect from ('172.21.40.6', 42668) handled by thread with name ThreadPoolExecutor-0_3 and id 140375016593088
python-z40_client_py-1 | Received 102400B of data
python-z40_client_py-1 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-2 exited with code 0
python-z40_client_py-1 exited with code 0
z40_server_container | Connect from ('172.21.40.7', 49844) handled by thread with name ThreadPoolExecutor-0_0 and id 140375116924608
python-z40_client_py-10 | Received 102400B of data
python-z40_client_py-10 | Client finished
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-10 exited with code 0
z40_server_container | Connect from ('172.21.40.8', 48846) handled by thread with name ThreadPoolExecutor-0_1 and id 140375033378496
z40_server_container | sending buffer # 1
z40_server_container | Connection closed by client
python-z40_client_py-7 | Received 102400B of data
python-z40_client_py-7 | Client finished
python-z40_client_py-7 exited with code 0

5.2.2 C



6 Wnioski

Obie implementacje potwierdzają prawidłowe działanie współbieżnego modelu serwera.
W implementacji w C połączenia są obsługiwane przez oddzielne procesy. Dzięki wykorzystaniu procesów każde połączenie działa w pełnej izolacji od innych, co zwiększa stabilność i bezpieczeństwo. Serwer w C może obsłużyć dowolną liczbę klientów, ograniczoną jedynie przez zasoby systemowe. Z tego powodu, każde połączenie zostało obsłużone przez proces z unikalnym PID.

W implementacji w Pythonie, serwer wykorzystuje pulę wątków, które są ponownie używane do obsługi kolejnych połączeń. Takie podejście jest bardziej efektywne pod względem zużycia zasobów, ponieważ wątki są lżejsze od procesów i wymagają mniej czasu na utworzenie oraz zarządzanie. Jednak wątki w Pythonie nie zapewniają pełnej izolacji między połączeniami, co może być wadą w bardziej złożonych systemach.