

# Programowanie Sieciowe

## Zadanie 2

Radosława Żukowska - Lider Zespołu

Aleksandra Szczypawka

Małgorzata Grzanka

27.11.2025r

Wersja sprawozdania: 1

### Zadanie 2 - Komunikacja TCP

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Klient oraz serwer musi być napisany w konfiguracji C + Python (do wyboru co w czym).

Zmodyfikować serwer tak, aby miał konstrukcję współbieżną, tj. obsługiwał każdego klienta w osobnym procesie. Dla C należy posłużyć się funkcjami `fork()` oraz (obowiązkowo) `wait()`. Dla Pythona należy posłużyć się wątkami, do wyboru: wariant podstawowy lub skorzystanie z `ThreadPoolExecutor`. Każdy połączony klient wysyła żądanie do serwera o obliczenie hasha przesłanej wiadomości. Przetestować dla kilku równolegle działających klientów.

### Rozwiążanie

Link do Repozytorium

Do realizacji zadania powstał:

- serwer TCP w języku C,
- klient TCP w Pythonie.

### Współbieżny serwer TCP

Działanie współbieżne serwera TCP zostało zapewnione za pomocą procesów. Dzięki temu przychodzące połączenia mogły zostać obsłużone współbieżnie. Główny proces uruchamia pętlę i nasłuchiwa na głównym gnieździe `listen(sock, ListenQueueSize)`; ale akceptuje połączenia już na `msgsock`. Następnie uruchamia podproces za pomocą `fork`, które kopiuje wszystkie zmienne głównego procesu. W podprocesie zostaje zamknięte gniazdo `sock` i zostaje przeprowadzone połączenie z klientem, przyjęcie danych, przesłanie odpowiedzi. Natomiast główny proces zamyka oryginalne gniazdo `msgsock` jako że jego kopia obsługuje dane w podprocesie i może wtedy przejść z powrotem do `accept()` aby przyjąć kolejne połączenie. Poniżej znajduje się fragment kodu implementującego opisane wyżej kroki.

```
while (1) {
    msgsock = accept(sock, (struct sockaddr *)0,
```

```

        (socklen_t *)0);
. . .
if (fork() == 0) {
    close(sock);
. . . // komunikacja z klientem
    close(msgsock);
    fflush(stdout);
    exit(0);
}
close(msgsock);
}
close(sock);

```

Aby usunąć procesy zombie została zaimplementowana następująca funkcja:

```

void sigchld_handler(int sig) {
    while (waitpid(-1, NULL, WNOHANG) > 0)
        ;
}

```

Serwer w tym zadaniu przyjmuje wiadomość od klienta i odpowiada na nią wyliczając jej hash. Wybrana została funkcja hashująca SHA256.

```
SHA256((unsigned char *)msg, msg_size, hash);
```

## Konfiguracja testowa

- Plik `tcp_client.py` implementuje jednego klienta TCP, natomiast wiele instancji tego programu zostaje uruchomionych za pomocą skryptu `entrypoint.sh`
- Każdy kontener w sieci `z36_network` dostaje dynamicznie przydzielony prywatny adres IP z podsieci Docker'a. Kontenery mogą komunikować się między sobą zarówno po tym adresie, jak i po aliasie nadanym przez `--network-alias`
- Serwer nasłuchiwa na porcie 8888. Klient łączy się na ten port.
- Komunikacja ograniczona do sieci Docker'a, więc opóźnienia sieci fizycznej praktycznie nie występują.

## Wyniki testów

Test został przeprowadzony poprzez uruchomienie serwera TCP oraz kilku klientów TCP. Każdy klient wysyłał ciąg 15 losowych znaków a serwer odpowiadał hashem. Na Rysunkach 1 i 2 zostały przedstawione uzyskane wyniki.

```
rzukowski@bigubu:~/PSI_lab_Z36/2$ sh testing_script.sh
--- SERVER LOGS ---
Server listening on port 8888
[PID 7] Connected
[PID 7] Client disconnected after sending data
[PID 7] Finished, hash sent
[PID 8] Connected
[PID 8] Client disconnected after sending data
[PID 9] Connected
[PID 9] Client disconnected after sending data
[PID 10] Connected
[PID 8] Finished, hash sent
[PID 10] Client disconnected after sending data
[PID 9] Finished, hash sent
[PID 11] Connected
[PID 11] Client disconnected after sending data
[PID 10] Finished, hash sent
[PID 12] Connected
[PID 12] Client disconnected after sending data
[PID 11] Finished, hash sent
[PID 12] Finished, hash sent
[PID 13] Connected
[PID 13] Client disconnected after sending data
[PID 13] Finished, hash sent
[PID 14] Connected
[PID 14] Client disconnected after sending data
[PID 14] Finished, hash sent
[PID 15] Connected
[PID 15] Client disconnected after sending data
[PID 16] Connected
[PID 16] Client disconnected after sending data
[PID 15] Finished, hash sent
[PID 17] Connected
[PID 17] Client disconnected after sending data
[PID 16] Finished, hash sent
[PID 17] Finished, hash sent
[PID 18] Connected
[PID 19] Connected
[PID 19] Client disconnected after sending data
[PID 18] Client disconnected after sending data
[PID 20] Connected
[PID 20] Client disconnected after sending data
[PID 19] Finished, hash sent
[PID 20] Finished, hash sent
[PID 18] Finished, hash sent
```

Rysunek 1: Informacje logujące od serwera.

```
rzukowski@bigubu:~/PSI_lab_Z36/2$ sh testing_script.sh
Client finished in 0.0019271159544587135 s. Message: DLJ3glzgW4phP0 Response: b'50a85373034b9666cca40ffccb14bf459cf73e7fcba794ced9da6db144ec9a'
Client finished in 0.0016891879495233296 s. Message: bkkbs750CLwi4386 Response: b'67608fbca980de50c52cc63f647ad9772a3b3f54bcce6cd8d14d2225ecb496'
Client finished in 0.0016557418275624514 s. Message: BcTu7jg0rATCjx8 Response: b'5b1d4ec98b33bc0619911677b9eb1340fe3a98511760d48dac7b57e591e94678'
Client finished in 0.0016632322219944 s. Message: jibWK60VsafhZc Response: b'd549005e4a0d7e7394e94ec77729fb7fb63682322ca36c081ff85e84dbac4791'
Client finished in 0.0013784330803900957 s. Message: 7RabF80Mb6jf1Y6 Response: b'fe64dffdd227abcd7bc05ee51f8764f0489922fce3cd7f0628385058c898a1406'
Client finished in 0.0011482189875096083 s. Message: 4Ew1g1G0CQTfIm7 Response: b'4a83da9c5105349efc56855d2e653048fb5ef976603e42e8393739d7afec4051'
Client finished in 0.0013850040268152952 s. Message: ERUZZP4P1ZU8sDY Response: b'6d0bca417f39cce695a183c2af73ca30970df944dc284e1de2f335b12b881'
Client finished in 0.0010947119444608688 s. Message: Qyb1P6SSQfnQsA0 Response: b'9672d4f4f01bfab5001b92e122f6f871094c62d43be61aaa18b66afa7400e86'
Client finished in 0.0012201869394630194 s. Message: Nyz0RSCBvefaiY Response: b'fe401698190a52e1ed1211dd31a9268c077b06cbefaae2cc83a4a6c0475ea74'
Client finished in 0.001387936031108284 s. Message: OwkzuAlaiQ8XrIr Response: b'493147e74e003584d78ef5deea52f60f4235c668d069347c95cd120b0e720df'
Client finished in 0.0013803441543132067 s. Message: v4mP0pGMdne1Lv Response: b'2d0124ff5f894c112cff32309c01e1074acf3cef22b37215e871ab2ad575cca'
Client finished in 0.0015197580214589834 s. Message: 08Dvnfm5Y9sV13x Response: b'fb1f7721d331b1943e15dc0ad74c467b87d4853f8684523f3bdd885a05544d2'
Client finished in 0.001768426038324833 s. Message: nhDRVvXN7fwLafK Response: b'a021f10671d6940fffb30386ea7bdaa347827edaf1fa09a2cd865b118eb0007b'
Client finished in 0.0016114229802042246 s. Message: wBsgZlAPiLomOCL Response: b'a8a13b804bccdb4790f285354cf3baef20e2ae4a16db4bdf1c8acd0a7accccfeb'
```

Rysunek 2: Informacje od klientów o czasie wykonania, wiadomości i odpowiedzi serwera (hash).

## **Wnioski i uwagi**

- Analizując logi serwera na Rysunku 1, można zobaczyć, że klienci nie są obsługiwani iteracyjnie, jeden po drugim. Występują sytuacje, w których połączenie z danym klientem nie zostało jeszcze zamknięte, a rozpoczyna się już komunikacja z innym klientem, w innym procesie. Świadczy to o współbieżnym działaniu serwera.
- Na Rysunku 2 jest pokazane, że czasy obsługi każdego z klientów są krótkie i zbliżone. Mierzenie czasu rozpoczęło się przed nawiązaniem komunikacji, a kończyło się po jej zakończeniu, więc pokazuje to, że klienci nie musieli czekać na bycie obsłużonym i byli obsługiwani współbieżnie.