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 kliku równolegle działających klientów.

Rozwiązanie

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 nasłuchuje 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.

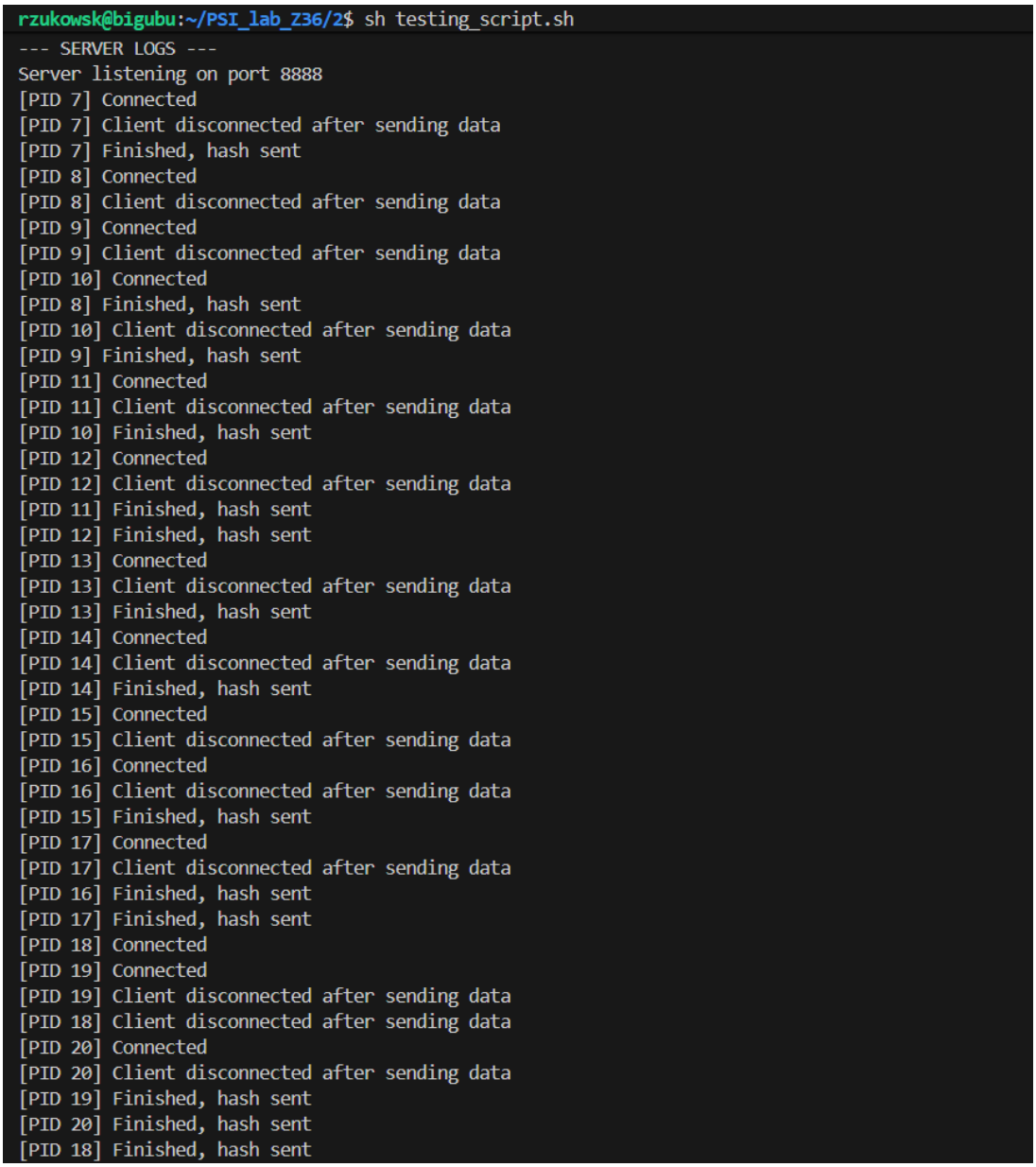
```
  SHA256((unsigned char *)msg, msg_size, hash);
```

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 Dockera. Kontenery mogą komunikować się między sobą zarówno po tym adresie, jak i po aliasie nadanym przez –network-alias
- Serwer nasłuchuje na porcie 8888. Klient łączy się na ten port.
- Komunikacja ograniczona do sieci Dockera, 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.



Rysunek 1: Informacje logujące od serwera.

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 rozpoczynało 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.