Programowanie Sieciowe

Laboratorium 2.1 + 2.2 wariant A

Sprawozdanie Zespołu 51

03.12.2025

Maciej Bogusławski 331362
Hubert Kaczyński 331386
Bartosz Żelazko 331457

1. **Opis zaimplementowanych programów**

Podczas realizacji zadania 2 sporządzone zostały po 2 programy dla obu zadań – klient C i serwer Python obsługujące połączenia TCP. Kod źródłowy został wykonany kierując się podstawowym kodem zmieszczonym przez prowadzącego przedmiot na serwerze studia. Kluczowym dla działania programów było zaprojektowanie przesyłanej struktury danych. Przyjęta została konstrukcja lisy jednokierunkowej, składającej się ze struktur „Node", w których pierwsze dwa bajty zarezerwowane są na 16 bitową liczbę całkowitą, kolejne cztery bajty na 32 bitową liczbę całkowitą i kolejne 64 bajty na zmienny ciąg znaków.

Na potrzebę laboratorium drugiego programy zostały dla przejrzystości rozdzielone dla zadań 2.1 i 2.2 w przeciwieństwie do pierwszych laboratoriów, gdzie odpowiednie argumenty wywołania mogły wykonać zadanie 1.1 lub 1.2.

Programy 2.1:

Serwer Python (ZAD2_1/p_server/p_server.py) po uruchomieniu wyczekuje na wiadomość na ustalonym porcie. Po nawiązaniu połączenia z klientem wypisywany na terminal jest jego adres, a przesyłana wiadomość jest odpowiednio odpakowywana i również wypisywana.
Przykładowe wywołanie klienta Python:
p_server.py --host 0.0.0.0 --port 8000
Serwer przyjmuje argument z adresem hosta oraz numerem używanego portu. Program posiada zaimplementowaną obsługę błędów funkcji systemowych i złego użycia argumentów wywołania.

Klient C (ZAD2_1/c_client/c_client.c) przyjmuje następujące wymagane argumenty wywołania:
- Adres połączenia,
- Port, który ma zostać wykorzystany,
- Ilość węzłów listy jednokierunkowej która ma zostać wysłana.

Program posiada zaimplementowaną obsługę błędów funkcji systemowych i złego użycia argumentów wywołania. Klient informuje o błędach i wykonanych czynnościach w terminalu.
Przykładowe wywołanie:
./c_client 0.0.0.0 8000 5

Do wykonania polecenia 2.1 zostały przygotowane odpowiednie pliki dockerowe, które odpowiadają za uruchomienie serwera i klienta. Programy działają poprawnie na serwerze Bigubu, ale ze względu na potrzebę użycia Asciinema do dokumentacji, nagrane eksperymenty wykonywane są w środowisku lokalnym. Nagrania terminala znajdują się w plikach zad2_1_client.cast oraz zad2_1_server.cast.

Programy 2.2:

serwer Python (ZAD2_2/p_server/p_server.py) po uruchomieniu wyczekuje na wiadomość na ustalonym porcie. Po nawiązaniu połączenia z klientem wypisywany na terminal jest jego adres, a przesyłana wiadomość jest odpowiednio odpakowywana i również wypisywana. Serwer wywołuje sztuczne uśpienie po odebraniu porcji danych z pomocą funkcji recv().
Przykładowe wywołanie klienta Python:
python3 -u p_server.py --host 0.0.0.0 --port 8000

Serwer przyjmuje argument z adresem hosta oraz numerem używanego portu. Program posiada zaimplementowaną obsługę błędów funkcji systemowych i złego użycia argumentów wywołania.

Klient C (ZAD2_2/c_client/c_client.c) przyjmuje następujące kolejne wymagane argumenty wywołania:
- Adres połączenia,
- Port, który ma zostać wykorzystany,
- Ilość węzłów listy jednokierunkowej która ma zostać wysłana,
- Wielkość bufora nadawczego w bajtach.

Klient w zadaniu 2.2 został zmodyfikowany tak aby dla zadanej ilości węzłów i wielkości bufora nadawczego wysyłał strumień danych do serwera, jednocześnie mierząc czas wysłania. Zmierzony wynik jest wliczany do statystyk i wyświetlany w terminalu. Na koniec połączenia prezentowane jest podsumowanie pomiarów. Program posiada zaimplementowaną obsługę błędów funkcji systemowych i złego użycia argumentów wywołania. Klient informuje o błędach i wykonanych czynnościach w terminalu.
Przykładowe wywołanie:
./c_client 0.0.0.0 8000 10000 100

Do wykonania polecenia 2.2 zostały przygotowane odpowiednie pliki dockerowe, które odpowiadają za uruchomienie serwera i klienta. Programy działają poprawnie na serwerze Bigubu, ale ze względu na potrzebę użycia Asciinema do dokumentacji, nagrane eksperymenty wykonywane są w środowisku lokalnym. Nagrania terminala znajdują się w plikach zad2_2_clients.cast i zad_2_2_server.cast.

2. Prezentacja działania programów i wnioski

W przypadku zadania 2.1 połączenie TCP odbywa się pomyślnie a serwer poprawnie odczytuje przesyłane dane. Wszystkie wymagane informacje są wypisywane w terminalu

Programy realizujące polecenie 2.2 poza udanym połączeniem TCP, pozwoliły na zaobserwowanie zjawiska, w którym odbiorca nie nadąża za szybkim nadawcą. Kod z zadania 2.1 został zmodyfikowany tak aby klient wysyłał duży strumień danych w pętli, a serwer symulował sztuczne opóźnienie przy użyciu krótkiego uśpienia. Zjawisko to wynika z natury połączenia TCP – w ramach kontroli przepływu danych protokół wstrzymuje nadawcę, gdy odbiorca nie nadąża z ich konsumpcją. Rozmiar bufora nadawczego nie miał istotnego wpływu na wyniki opóźnień. Bierze się to z faktu, że rozmiar wysyłanych danych i tak zawsze przekraczał rozmiar bufora nadawczego więc wstrzymywanie wysyłania dla wszystkich przypadków zachowywało się podobnie.

Poniżej znajdują się wyniki dla przeprowadzonych eksperymentów:
Wysłane węzły: 10000 po 70 bajtów
Sztuczne opóźnienie: 0,01s
Jako „wolne” wysłania uznano takie, które trwają dłużej niż 0.1 milisekundy.

Dla bufora nadawczego 100B:
Maksymalny czas wysyłki węzła: 762,574140 ms
Minimalny czas wysyłki węzła: 0,000180 ms
Średni czas wysyłki węzła: 0,601278 ms

Ilość „wolnych” wysłań: 17

Dla bufora nadawczego 1KB:
Maksymalny czas wysyłki węzła: 762,678180 ms
Minimalny czas wysyłki węzła: 0,000180 ms
Średni czas wysyłki węzła: 0,603471 ms
Ilość „wolnych” wysłań: 19

Dla bufora nadawczego 10KB:
Maksymalny czas wysyłki węzła: 897,434100 ms
Minimalny czas wysyłki węzła: 0,000180 ms
Średni czas wysyłki węzła: 0,615935 ms
Ilość „wolnych” wysłań: 17