PSI zad 2 - sprawozdanie

02.12.2024

Zespół Z31:

- Marcin Bagnowski
- Maciej Kaniewski
- Tomasz Kurzela

Repozytorium GitHub: https://github.com/Kapturz0ny/PSI_Z31

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.

Klient TCP wysyła złożoną strukturę danych. Przykładowo: tworzymy w pamięci listę jednokierunkową lub drzewo binarne struktur zawierających (oprócz danych organizacyjnych) pewne dane dodatkowe: np. liczbę całkowitą 16-o bitową, liczbę całkowitą 32-u bitową oraz napis zmiennej i ograniczonej długości. Serwer napisany w Pythonie/C powinien te dane odebrać, dokonać poprawnego „odpakowania” tej struktury i wydrukować jej pola (być może w skróconej postaci, aby uniknąć nadmiaru wyświetlanych danych). Klient oraz serwer powinny być napisane w różnych językach. Wskazówka: można wykorzystać moduły Python-a: struct i io.

Rozwiązanie

Opis struktury

Korzystamy z listy jednokierunkowej, przechowuje ona elementy o strukturze:

- id, 4B - identyfikator, nie musi być niepowtarzalny
- charlen, 2B - długość napisu (*string*)
- string, <charlen>B - napis

Długość napisu (*charlen*) jest losowana przy tworzeniu elementu listy z zakresu [5, 50]. W naszym przykładzie napis jest generowany podobnie jak w zadaniu 1 - kolejne litery alfabetu, powtarzające się cyklycznie.

Opis konfiguracji testowej

Sieć: docker network o nazwie `z31_network`

Nie deklarujemy adresów IP wprost, korzystamy z nazw kontenerów i resolvera DNS w programach.

Serwer nasłuchuje na porcie 8000

Programy:

- klient napisany w C
- serwer napisany w Python

Wykonanie

Domyślnie klient wysyła listę o rozmiarze 100_000 elementów, przez co na pewno spełniony jest warunek przesłania 100kB.

Wyświetlane jest zarówno pierwsze i ostatnie 10 elementów, żeby zachować przejrzystość.

Uruchomienie

Jednolinijkowe

`docker compose up --build` buduje, a następnie uruchamia wszystkie kontenery

Zalecane jest jednak wcześniejsze zbudowanie wszystkich kontenerów poleceniem `docker compose build` Następnie można uruchomić w oddzielnych terminalach:

- `docker compose up z31_tcp_server` serwer Python
- `docker compose up z31_zad2_client` klient C

Testowanie

```
tkurzela@bigubu:~/PSI_Z31/zad2$ docker compose up --build
[+] Building 9.3s (16/16) FINISHED
(...)
Attaching to z31_tcp_server, z31_zad2_client
z31_tcp_server   | Server started on 0.0.0.0:8000
z31_tcp_server   | Waiting for a connection...
z31_tcp_server   | Connected by ('172.21.31.3', 45686)
z31_zad2_client  | C client for zadanie 2
z31_zad2_client  | Will send to z31_tcp_server:8000
z31_zad2_client  | id:00000, len:05 : ABCDE
z31_zad2_client  | id:00001, len:33 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG
z31_zad2_client  | id:00002, len:21 : ABCDEFGHIJKLMNOPQRSTU
z31_zad2_client  | id:00003, len:32 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
z31_zad2_client  | id:00004, len:30 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCD
z31_zad2_client  | id:00005, len:28 : ABCDEFGHIJKLMNOPQRSTUVWXYZAB
z31_zad2_client  | id:00006, len:30 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCD
z31_zad2_client  | id:00007, len:06 : ABCDEF
z31_zad2_client  | id:00008, len:17 : ABCDEFGHIJKLMNOPQ
z31_zad2_client  | id:00009, len:29 : ABCDEFGHIJKLMNOPQRSTUVWXYZABC
z31_zad2_client  | id:00010, len:06 : ABCDEF
z31_zad2_client  | id:99990, len:40 :
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
z31_zad2_client  | id:99991, len:36 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ
z31_zad2_client  | id:99992, len:12 : ABCDEFGHIJKL
```

```
z31_zad2_client  | id:99993, len:17 : ABCDEFGHIJKLMNOPQ
z31_zad2_client  | id:99994, len:18 : ABCDEFGHIJKLMNOPQR
z31_zad2_client  | id:99995, len:07 : ABCDEFG
z31_zad2_client  | id:99996, len:46 :
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
z31_zad2_client  | id:99997, len:32 : ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
z31_zad2_client  | id:99998, len:08 : ABCDEFGH
z31_zad2_client  | id:99999, len:28 : ABCDEFGHIJKLMNOPQRSTUVWXYZAB
z31_zad2_client  | Connected to the server.
z31_zad2_client  | Succesfully sent 3300931 bytes of data
z31_zad2_client  | Sent list with 100000 nodes
z31_tcp_server   | Closing connection with ('172.21.31.3', 45686)
z31_tcp_server   | id:00000, len:05 : b'ABCDE'
z31_tcp_server   | id:00001, len:33 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFG'
z31_tcp_server   | id:00002, len:21 : b'ABCDEFGHIJKLMNOPQRSTU'
z31_tcp_server   | id:00003, len:32 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF'
z31_tcp_server   | id:00004, len:30 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCD'
z31_tcp_server   | id:00005, len:28 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZAB'
z31_tcp_server   | id:00006, len:30 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCD'
z31_tcp_server   | id:00007, len:06 : b'ABCDEF'
z31_tcp_server   | id:00008, len:17 : b'ABCDEFGHIJKLMNOPQ'
z31_tcp_server   | id:00009, len:29 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABC'
z31_tcp_server   | id:00010, len:06 : b'ABCDEF'
z31_tcp_server   | id:99990, len:40 :
b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN'
z31_tcp_server   | id:99991, len:36 :
b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ'
z31_tcp_server   | id:99992, len:12 : b'ABCDEFGHIJKL'
z31_tcp_server   | id:99993, len:17 : b'ABCDEFGHIJKLMNOPQ'
z31_tcp_server   | id:99994, len:18 : b'ABCDEFGHIJKLMNOPQR'
z31_tcp_server   | id:99995, len:07 : b'ABCDEFG'
z31_tcp_server   | id:99996, len:46 :
b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST'
z31_tcp_server   | id:99997, len:32 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF'
z31_tcp_server   | id:99998, len:08 : b'ABCDEFGH'
z31_tcp_server   | id:99999, len:28 : b'ABCDEFGHIJKLMNOPQRSTUVWXYZAB'
z31_tcp_server   | Received 3300931 bytes of data
z31_tcp_server   | Node count:  100000
z31_zad2_client exited with code 0
```

Wnioski

Komunikacja protokołem TCP gwarantuje niezawodne dostarczenie danych w odpowiedniej kolejności. Zrealizowana komunikacja pokazała, że TCP pozwala efektywnie przesyłać duże struktury danych, takie jak lista jednokierunkowa, w sposób przewidywalny i bezbłędny.

W porównaniu do zadań realizowanych wcześniej (z użyciem UDP), TCP zapewnia stabilność poprzez utworzenie połączenia wraz z automatyczną retransmisją zgubionych pakietów. Dla UDP taką retransmisję musieliśmy samodzielnie zapewnić. Dzięki swoim mechanizmom TCP jest idealnym wyborem w przypadkach wymagających niezawodności, takich jak transmisja plików, bazy danych czy komunikatory tekstowe.