PSI - Zadanie 2

02.12.2024

Zespół Z39 w składzie:
Adrian Murawski
Kacper Straszak
Michał Brzeziński

Komunikacja TCP

Treść zadania

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

- utworzony został klient w języku C, przesyłający drzewo binarne do serwera napisanego w pythonie

klient

- plik tcp_client.c wysyłający drzewo serwerowi zawierający
    - funkcja resolve_host - pozwalający na korzystanie z aliasu hosta zamiast jego adresu IP
    - Struktura TreeNode - implementacja węzła drzewa od strony klienta
    - funkcja serialize_tree - funkcja konwertująca postać drzewa na reprezentację binarną
    - funkcja randint - do określenia losowo dobranej długości tekstu dla węzłów drzewa
    - funkcja create_sample_tree - tworząca przykładowe drzewo i zwracająca jego pointer
    - funkcja start_client - jest odpowiedzialna za połączenie z serwerem i wysłaniem mu sygnału

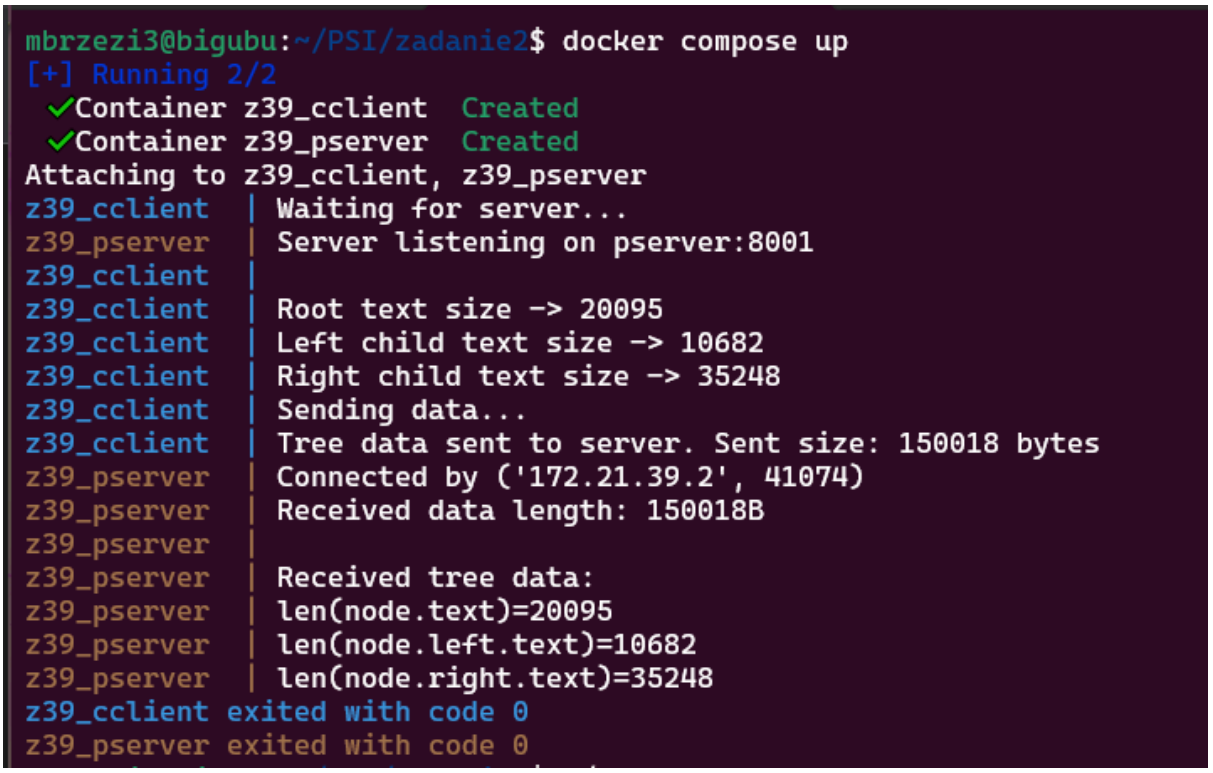
serwer

- plik tcp_server.py zawierający kod rozpoczynający pracę serwera i oczekujący na otrzymanie i zbierający dane wysłane przez klienta

- powstał plik pomocniczy dla serwera, node.py zawierający potrzebne klasy i metody
    - klasa TreeNode - reprezentujące węzły drzewa zawierające różne rodzaje zmiennych takie jak data_16 (2-bajtowa licza całkowita), czy text, odzwierciedlająca strukturę drzewa TreeNode w kodzie klienta
    - funkcja unpack_data - funkcja odwrotna do serialize_tree u klienta. Odtwarza węzeł drzewa z danych binarnych
    - deserialize tree - odtwarza drzewo binarne z danych binarnych

Wnioski

Przykładowe uruchomienie:



```
mbrzezi3@bigubu:~/PSI/zadanie2$ docker compose up
[+] Running 2/2
 ✔Container z39_pserver  Created
 ✔Container z39_cclient  Created
Attaching to z39_cclient, z39_pserver
z39_cclient  | Waiting for server...
z39_pserver  | Server listening on pserver:8001
z39_cclient  |
z39_cclient  | Root text size -> 14680
z39_cclient  | Left child text size -> 13199
z39_cclient  | Right child text size -> 14311
z39_cclient  | Sending data...
z39_cclient  | Tree data sent to server. Sent size: 150018 bytes
z39_pserver  | Connected by ('172.21.39.2', 40584)
z39_pserver  | Received data length: 150018B
z39_pserver  |
z39_pserver  | Received tree data:
z39_pserver  | len(node.text)=14680
z39_pserver  | len(node.left.text)=13199
z39_pserver  | len(node.right.text)=14311
z39_cclient exited with code 0
z39_pserver exited with code 0
```

- Jak można zauważyć na podstawie różnych uruchomień, funkcja generuje drzewa o losowej długości tekstu
- dane odebrane przez serwer mają tą samą wielkość
- pakiety pozostają tego samego rozmiaru pomimo różnej długości tekstu ponieważ zaalokowane jest zawsze pole o długości TEXTFIELD_SIZE

Natrafione problemy

- Po natrafieniu na problem związanym z ograniczeniem wielkości przesyłanych danych potrzebowaliśmy zwiększyć limit na ponad 100kB za pomocą metody

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 151 * 1024)
```