PSI - Sprawozdanie zad 2

Autorzy:
Weronika Maślana
Alesia Filinkova
Diana Pelin

Z 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). Klient wysyła złożoną strukturę danych w postaci idealnego drzewa binarnego (co najmniej 15 węzłów). Każdy węzeł zawiera (oprócz danych organizacyjnych) liczbę całkowitą. Serwer powinien te dane odebrać (jeden węzeł drzewa na wiadomość) oraz dokonać poprawnej rekonstrukcji. Wskazówka: można wykorzystać moduły Python-a: struct i io.

1. Opis rozwiązania problemu

Celem zadania była implementacja komunikacji TCP pomiędzy dwoma programami – klientem napisanym w Pythonie oraz serwerem napisanym w języku C. Przesyłaną strukturą danych jest idealne drzewo binarne składające się z co najmniej 15 węzłów, gdzie każdy węzeł zawiera identyfikator, wartość całkowitą oraz numery lewego i prawego dziecka. Klient wysyła kolejne węzły drzewa jako osobne wiadomości (po jednej linii na węzeł), natomiast serwer odbiera je, parsuje zawartość i rekonstruuje pełną strukturę drzewa w pamięci

a) Klient Python

   Klient jest odpowiedzialny za utworzenie kompletnej struktury idealnego drzewa binarnego. Węzły numerowane są od 0 do 14, zgodnie z typowym sposobem reprezentacji drzewa binarnego w tablicy

   - lewe dziecko: 2*id+1
   - prawe dziecko: 2*id+2
   - jeśli indeks przekracza liczbę węzłów, przypisywana jest wartość -1

b) Serwer C

   Serwer nasłuchuje na porcie TCP 5000. Po przyjęciu połączenia odbiera dane w sposób strumieniowy, czytając pojedyncze bajty funkcją recv()
   Informacje o każdym węźle są tymczasowo przechowywane w:

   - tablicy struktur nodes[] (wartość i wskaźniki)
   - dwóch pomocniczych tablicach left_ids[] oraz right_ids[]

Po zakończeniu odbioru danych serwer rekonstruuje pełne drzewo poprzez uzupełnienie wskaźników left oraz right

c) Prezentacja wyniku (preorder) i zapis do pliku

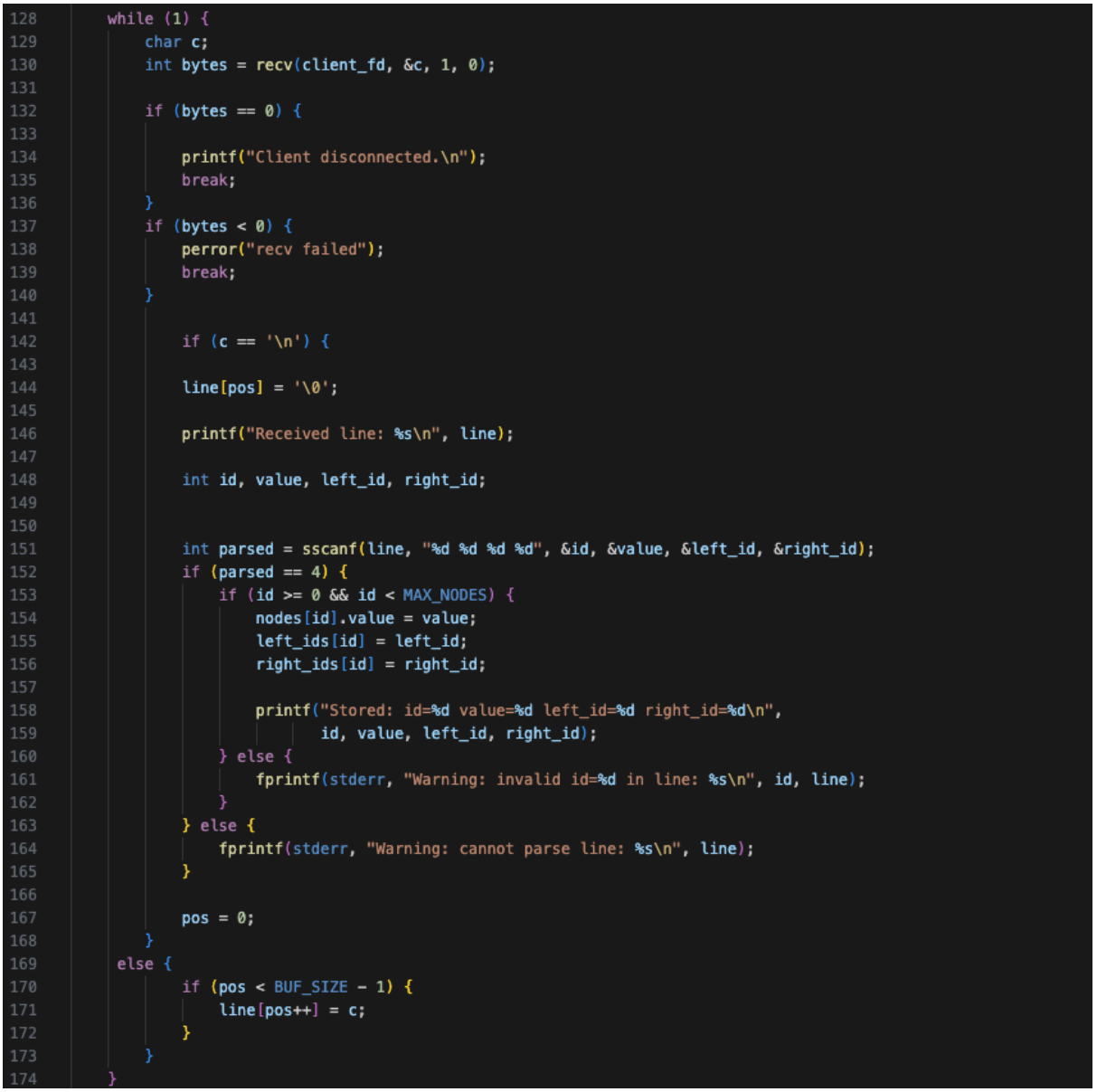
Po odtworzeniu drzewa serwer wykonuje przejście preorder i wypisuje całą strukturę w kolejności: korzeń -> lewe poddrzewo -> prawe poddrzewo

Wynik jest:

- wyświetlany w konsoli (logi),
- zapisywany do pliku tree_preorder.txt w kontenerze Dockera

```
47  def send_tree(nodes):
48      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
49
50      s.connect((HOST, PORT))
51      print("Connected, sending nodes")
52
53      for node in nodes:
54          line = f"{node.node_id} {node.value} {node.left_id} {node.right_id}\n"
55          data = line.encode("utf-8")
56
57          print(f"sending: {line.strip()}")
58          s.sendall(data)
59
60      print("All nodes sent, closing socket")
61      s.close()
```

Wysyłanie drzewa po stronie klienta



```c
while (1) {
    char c;
    int bytes = recv(client_fd, &c, 1, 0);

    if (bytes == 0) {

        printf("Client disconnected.\n");
        break;
    }
    if (bytes < 0) {
        perror("recv failed");
        break;
    }

        if (c == '\n') {

        line[pos] = '\0';

        printf("Received line: %s\n", line);

        int id, value, left_id, right_id;


        int parsed = sscanf(line, "%d %d %d %d", &id, &value, &left_id, &right_id);
        if (parsed == 4) {
            if (id >= 0 && id < MAX_NODES) {
                nodes[id].value = value;
                left_ids[id] = left_id;
                right_ids[id] = right_id;

                printf("Stored: id=%d value=%d left_id=%d right_id=%d\n",
                       id, value, left_id, right_id);
            } else {
                fprintf(stderr, "Warning: invalid id=%d in line: %s\n", id, line);
            }
        } else {
            fprintf(stderr, "Warning: cannot parse line: %s\n", line);
        }

        pos = 0;
    }
 else {
        if (pos < BUF_SIZE - 1) {
            line[pos++] = c;
        }
    }
}
```

Odebranie danych po stronie serwera

```
void linkTree() {
    for (int i = 0; i < MAX_NODES; i++) {
        if (left_ids[i] != -1) {
            nodes[i].left = &nodes[left_ids[i]];
        } else {
            nodes[i].left = NULL;
        }

        if (right_ids[i] != -1) {
            nodes[i].right = &nodes[right_ids[i]];
        } else {
            nodes[i].right = NULL;
        }
    }
}
```

Odtworzenie poprawnej struktury drzewa

2. Opis konfiguracji testowej

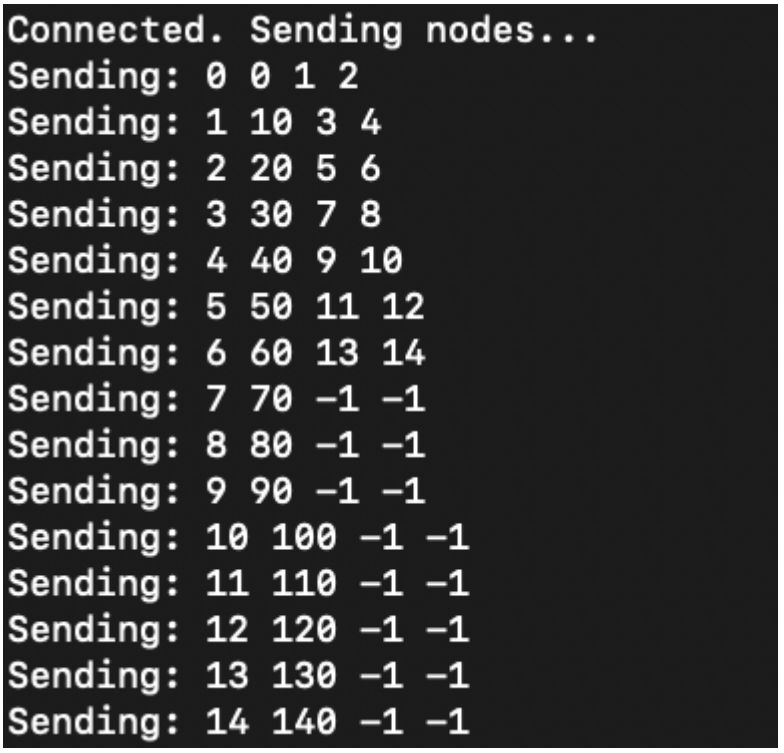
a) Do testów rozwiązania wykorzystano dwa kontenery Docker uruchomione na serwerze bigubu i połączone we wspólnej sieci Docker z31_network, zgodnie z wymaganiami środowiska laboratoryjnego.

- Serwer TCP
  - nazwa kontenera: z31_server2
  - obraz: z31_server2
  - język: C (gcc)
  - port nasłuchujący: 5000/TCP
  - sieć: z31_network

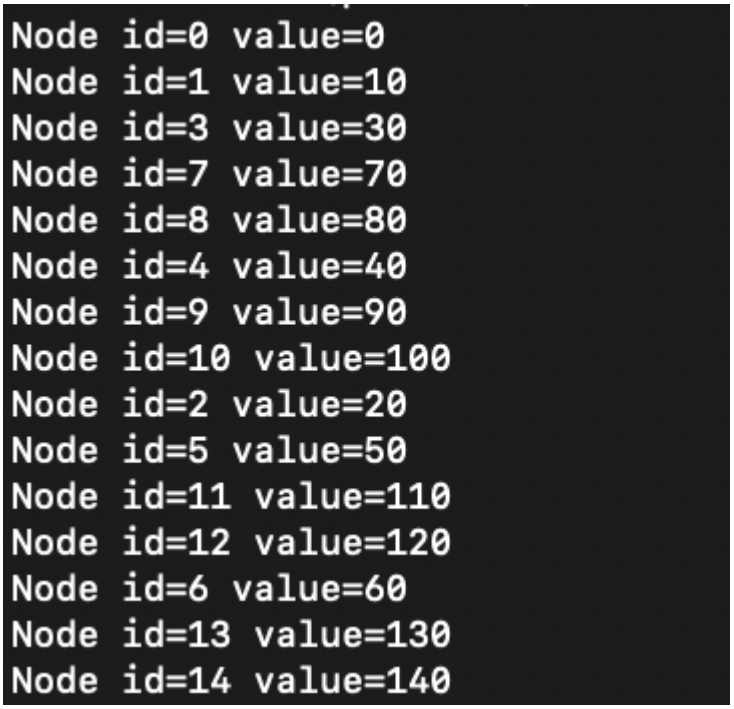
- Klient TCP
  - nazwa kontenera: z31_client2
  - obraz: z31_client2
  - język: Python 3
  - sieć: z31_network
  - serwer dostępny pod hostname: z31_server2

3. Opis wyników

Serwer poprawnie odebrał i dokonał poprawnej rekonstrukcji



Wiadomość wysłana przez klienta



Wiadomość odtworzona przez serwer

4. Wnioski

Zadanie pokazało, jak działa komunikacja TCP i jak przesyłać dane między dwoma programami w różnych językach. Udało się poprawnie wysłać drzewo binarne z Pythona do serwera w C i je tam odtworzyć. Wszystko działa w Dockerze, więc rozwiązanie spełnia wymagania i pozwala na łatwe testowanie.