PSI Zadanie 2

Paweł Spirydowicz, Hanna Zarzycka, Krzysztof Wojtaszko
27.11.2025 wersja 1

Treść zadania

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.

Rozwiązanie

Rozwiązanie składa się z dwóch komponentów, klienta i serwera, umieszczonych w kontenerach docker. Klient i serwer są podłączone do tego samego kontenera z siecią, dzięki czemu mogą przesyłać między sobą pakiety. Zarówno dla klienta jak i serwera zdefiniowana jest struktura węzłów, które składają się na drzewo binarne.

Klient C

Drzewo

Węzeł drzewa

```
typedef struct Node {
    int id;
    int value;
    struct Node* left;
    struct Node* right;
} Node;
```

Tworzenie węzła

```
Node* createNode(int id, int value)
{
    Node* node = malloc(sizeof(Node));
    node->id = id;
    node->value = value;
    node->left = NULL;
```

```
    node->right = NULL;
    return node;
}
```

Tworzenie dzieci i przyłączanie ich do drzewa

```
Node* makeChild(int index, int n)
{
    if (index >= n)
        return NULL;

    Node* node = createNode(index, index);
    node->left  = makeChild(2*index + 1, n);
    node->right = makeChild(2*index + 2, n);

    return node;
}
```

Tworzenie drzewa o podanej ilości węzłów

```
Node* makeTree(int n)
{
    if (n <= 0) return NULL;
    return makeChild(0, n);
}
```

Usuwanie drzewa

```
void freeTree(Node* tree)
{
    if (!tree) return;
    freeTree(tree->left);
    freeTree(tree->right);
    free(tree);
}
```

Komunikacja TCP

Przygotowanie komunikacji z serwerem TCP

```
    // Initialize server struct
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;

    server.sin_port = htons(atoi(argv[2]));

    //IP from name
    server.sin_family = AF_INET;
    hp = gethostbyname2(argv[1], AF_INET );
```

```
    if (hp == (struct hostent *) 0) {
        errx(2, "%s: unknown host\n", argv[1]);
    }
    printf("Address resolved\n");

    memcpy((char *) &server.sin_addr, (char *) hp->h_addr, hp->h_length);

    // Connect to server
    connect (sock, (struct sockaddr *) &server, sizeof(server) );
```

Przesłanie drzewa

```
void sendBinaryTree(Node* tree, int sock, int treeSize)
{
    if (!tree) return;

    Node** queue = malloc(sizeof(Node*) * treeSize);
    int front = 0, back = 0;

    queue[back++] = tree;

    while (front < back)
    {
        Node* n = queue[front++];

        int data[2] = { n->id, n->value };
        if (send(sock, data, sizeof(data), 0) != sizeof(data))
            bailout("send failed");

        if (n->left)  queue[back++] = n->left;
        if (n->right) queue[back++] = n->right;
    }
    free(queue);
}
```

Drzewo jest przesyłane po jednym węźle na strumień w kolejności zgodnej z algorytmem przeszukiwania w głąb. Dane wierzchołka przesyłane do serwera TCP to id węzła i liczba, która się w nim znajduje.

Zamknięcie gniazda

```
close(sock);
```

Serwer Python

Drzewo

Węzeł drzewa

```
class Node:
```

```
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

Odbudowywanie drzewa z danych od klienta

```
def build_tree(values):
    if values[0] is None:
        return None
    root = Node(values[0])
    queue = [(root, 0)]
    while queue:
        node, idx = queue.pop(0)
        li = 2 * idx + 1
        ri = 2 * idx + 2
        if li < len(values) and values[li] is not None:
            node.left = Node(values[li])
            queue.append((node.left, li))
        if ri < len(values) and values[ri] is not None:
            node.right = Node(values[ri])
            queue.append((node.right, ri))
    return root
```

Odbudowa drzewa odbywa się na tej samej zasadzie, na której zostało wysłane - zgodnie z działaniem algorytmu przeszukiwania w głąb, chociaż w tym przypadku nie jest to przeszukiwanie, a dodawanie węzłów w uporządkowanej kolejności.

Komunikacja TCP

Konfiguracja serwera i odebranie połączenia od klienta

```
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(1)
    print(f"Serwer nasłuchuje na {HOST}:{PORT}")

    conn, addr = server.accept()
    print("Połączono z:", addr)
```

Odbieranie danych ze strumienia

```
def recv_exact(conn, n):
    data = b""
    while len(data) < n:
        chunk = conn.recv(n - len(data))
        if not chunk:
            return None
        data += chunk
    return data
```

```
values = [None] * TREE_SIZE
for _ in range(TREE_SIZE):
    packet = recv_exact(conn, 8)
    if packet is None:
        break
    node_id, value = struct.unpack("<ii", packet)
    values[node_id] = value
    print(f"Odebrano: id={node_id}, value={value}")
```

Zamknięcie połączenia i gniazda

```
conn.close()
server.close()
```

Konfiguracja docker

Serwer - z37_server
Klient - z37_client
Sieć - z37_network

Wywołanie programów

app2.sh

Testy

Testujemy program poprzez sprawdzenie czy serwer odbiera i poprawnie rekonstruuje dane.

Konfiguracja

Port 8000
IP przydzielone automatycznie

Odebrane dane

Connected to: z37_server
Połączono z: ('172.21.37.3', 34884)
Odebrano: id=0, value=0
Odebrano: id=1, value=1
Odebrano: id=2, value=2
Odebrano: id=3, value=3
Odebrano: id=4, value=4
Odebrano: id=5, value=5
Odebrano: id=6, value=6
Odebrano: id=7, value=7
Odebrano: id=8, value=8
Odebrano: id=9, value=9

Odebrano: id=10, value=10
Odebrano: id=11, value=11
Odebrano: id=12, value=12
Odebrano: id=13, value=13
Odebrano: id=14, value=14

Rekonstrukcja drzewa

Drzewo:
Poziom 0: 0
Poziom 1: 1 2
Poziom 2: 3 4 5 6
Poziom 3: 7 8 9 10 11 12 13 14

Wnioski

Implementacja przesyłu danych po stronie klienta protokołem TCP różni się od UDP głównie tym, że odbywa się poprzez strumień, a nie datagram. Funkcja connect() ma też inne działanie, bo zgodnie z nazwą nawiązuje połączenie z serwerem, gdy w przypadku UDP jedynie zapamiętywała domyślny adres. Podczas implementacji serwera należało użyć funkcji listen, która umożliwia nasłuchiwanie połączeń oraz accept do ich akceptacji. Samo odbieranie danych ze strumienia wymagało specyficznej implementacji, ponieważ dane są podzielone na kawałki.

Rekonstrukcja drzewa binarnego była możliwa dzięki ustaleniu wspólnej metody przeszukiwania między serwerem a klientem. Drzewo zostało wysłane po jednym węźle w postaci prostej dwuelementowej tablicy. Po stronie serwera napisanego w pythonie można było ją rozpakować z pomocą biblioteki struct.