PSI laboratorium Z 2

Lider - Miłosz Andryszczuk
Aleksander Paliwoda
Maksymilian Zieliński

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.

Z 2:

Klient napisany w języku C tworzy idealne drzewo binarne składające się z 15 węzłów. Następnie przechodzi po nim w porządku pre-order i wysyła po jednym pakiecie na każdy odwiedzony węzeł.

Każdy pakiet to struktura zawierająca dwie liczby 32-bitowe: indeks oraz wartość węzła. Aby zapewnić zgodność przesyłanych danych, struktura została spakowana za pomocą dyrektywy #pragma pack(push, 1), a liczby konwertowane są do sieciowej kolejności bajtów (Big-Endian) za pomocą funkcji htonl.

Fragment kodu klienta odpowiedzialny za wysyłanie pakietów:

```
void sendTree(Node *node, int index, int sock) {
  if (node == NULL)
    return;
  Packet pkt;
  pkt.index = htonl(index);
  pkt.value = htonl(node->value);
```

```
  if (send(sock, &pkt, sizeof(Packet), 0) < 0) {
    perror("send failed");
    return;
  }

  printf("Sent node: index=%d, value=%d\n", ntohl(pkt.index),
ntohl(pkt.value));
  sendTree(node->left, 2 * index + 1, sock);
  sendTree(node->right, 2 * index + 2, sock);
}
```

Serwer w języku Python nasłuchuje na porcie TCP. Serwer odbiera dane w pętli, każdorazowo oczekując dokładnie 8 bajtów. Odebrane dane są deserializowane przy użyciu struct.unpack z formatem !ii.

Węzły są buforowane w słowniku nodes, gdzie kluczem jest indeks. Po zakończeniu transmisji serwer rekonstruuje pełną strukturę drzewa, łącząc wskaźniki left i right na podstawie matematycznej zależności indeksów (2*i + 1, 2*i + 2).

Fragment kodu serwera odpowiedzialny za odbiór pakietów:

```
while True:
      data = conn.recv(PACKET_SIZE)
      if not data:
            break

      if len(data) == PACKET_SIZE:
            index, value = struct.unpack(PACKET_FORMAT, data)
            print(f"Received data: index={index}, value={value}")
            nodes[index] = Node(value, index)
      else:
            print(f"Received incomplete packet! ({len(data)} bytes)")
```

Opis konfiguracji testowej:

Compose:

- Z34_tcp_server - kontener serwera,
- Z34_tcp_client - kontener klienta,
- Z34_network - sieć mostkowa Docker.

Opis testowania:

Klient uruchamiany był z poziomu Docker Compose i automatycznie nawiązywał połączenie z serwerem. Program klienta wygenerował drzewo, zserializował je i wysłał kolejno 15 pakietów.

Po stronie serwera odebrano wszystkie pakiety bez błędów. Serwer zbuforował węzły, a po rozłączeniu się klienta poprawnie odbudował strukturę drzewa. Weryfikacja poprawności polegała na wyświetleniu przez serwer drzewa w porządku Pre-order.

Wnioski końcowe:

- Zastosowanie TCP znacząco uprościło kwestię niezawodności transmisji oraz kolejności pakietów.
- Najbardziej krytycznym elementem implementacji jest konieczność prawidłowej serializacji i deserializacji danych pomiędzy dwoma różnymi językami