17.11.2024

Zespół 36

Mateusz Łukasiewicz

Rafał Celiński

Przemysław Walecki

# Programowanie sieciowe

## Zadanie 1.1

**Z 1 Komunikacja UDP**

Napisz zestaw dwóch programów – klienta i serwera wysyłające datagramy UDP. Wykonaj ćwiczenie w kolejnych inkrementalnych wariantach (rozszerzając kod z poprzedniej wersji). Klient jak i serwer powinien być napisany zarówno w C jak i Pythonie (4 programy).

Sprawdzić i przetestować działanie „między-platformowe”, tj. klient w C z serwerem Python i vice versa.

### Konfiguracja

Adres IP serwera – 172.21.36.2

Adres IP klienta – 172.21.36.3

Porty – 5018:5018 (UDP)

### Obrazy Docker

Obraz dla Pythona - python:3-slim

Obraz dla GCC - gcc:4.9

## Rozwiązanie

Programy klienta i serwera zrealizowaliśmy w dwóch językach – C++ i Python. Zasada działania i struktura obu wersji jest taka sama.

### Klient

1. Tworzenie gniazda UDP do komunikacji z serwerem

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
```

2. Rozpoczęcie pętli, która będzie działać dopóki nie znajdziemy największej wielkości przesyłanego datagramu i tworzenie samego datagramu

```
while True:
    # Pierwsze 4 bajty reprezentują długość datagramu
    message = bytearray(message_size + len)
    message[:len] = message_size.to_bytes(len, byteorder='little')

    # W reszcie wiadomości ustawiamy powtarzające się literry alfabetu
    for i in range(len, message_size + len):
        message[i] = 65 + (i % 26)
```

3. Próba wysłania datagramu - w przypadku niepowodzenia zmniejszamy jego rozmiar o 1 aż nam się uda

```
try:
    sent_bytes = s.sendto(message, (HOST, UDP_PORT))
    print(f"Size of sent message: {message_size} bytes, whole datagram size: {message_size + len}")
    if(failed_send):
        found_max = True
except Exception as e:
    failed_send = True
    print(f"Couldn't send message of size: {message_size + len} bytes")
    message_size -= 1
    time.sleep(0.2)
    continue
```

4. W przypadku powodzenia czekamy na przysłanie potwierdzenia przez serwer

```
# Odebranie odpowiedzi
try:
    data, address = s.recvfrom(1024)
    if data:
        print(f"Received answer: {data.decode()}")
        if data.decode() != "OK":
            print("Invalid packet recevied on server side, sending message with same size.")
            time.sleep(0.5)
            continue
except socket.timeout:
    print("Timeout")
```

5. Gdy znajdziemy rozwiązanie zadania wypisujemy je i kończymy działanie klienta

```
    if(found_max):
        print(f"Maximum datagram size sent: {message_size + len}, message size: {message_size}")
        break

    message_size *= 2
    time.sleep(0.5)

s.close()
```

**Serwer**

1. Tworzenie gniazda UDP i powiązanie go z adresem IP i portem serwera aby móc nasłuchiwać na połączenia

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.bind((IP, UDP_PORT))
```

2. Odbieranie danych z gniazda UDP – specjalnie została użyta liczba 100000 jako rozmiar bufora, żeby na pewno odgórnie nie ograniczyć rozwiązania

```
    while True:
        data, addr = s.recvfrom(100000)
        print(f"Received datagram of size {len(data)} from {addr}")
```

3. Walidacja długości otrzymanej wiadomości i odesłanie klientowi odpowiedzi

```
        if message_size + 4 == len(data):
            print(f"Message length validated successfully for message of size {message_size} bytes.")
            s.sendto(b"OK", addr)
        else:
            print(f"Expected {message_size + 4} bytes, got {len(data)}.")
            s.sendto(b"FAIL", addr)
```

**Działanie programów**

Poza tym, że rozwiązania działają w parach serwer i klient ‘.py’ i ‘.cpp’ to możliwe jest krzyżowanie – tak jak było to w poleceniu.

Po zbudowaniu i uruchomieniu kontenerów w Dockerze uzyskaliśmy poniższe wyniki

```
[+] Running 2/2
 ✔Container z36_server_container  Created
 ✔Container z36_client_container  Recreated
Attaching to z36_client_container, z36_server_container
z36_server_container  | Server listening on 172.21.36.2:5018
z36_client_container  | Sent datagram of size 6 bytes, message size: 2
z36_server_container  | Received datagram of size 6 from ('172.21.36.3', 39777)
z36_server_container  | Message length validated successfully for message of size 2 bytes.
z36_client_container  | Received acknowledgment: OK
z36_client_container  | Sent datagram of size 8 bytes, message size: 4
z36_server_container  | Received datagram of size 8 from ('172.21.36.3', 39777)
z36_client_container  | Received acknowledgment: OK
z36_server_container  | Message length validated successfully for message of size 4 bytes.
z36_client_container  | Sent datagram of size 12 bytes, message size: 8
z36_server_container  | Received datagram of size 12 from ('172.21.36.3', 39777)
z36_server_container  | Message length validated successfully for message of size 8 bytes.
z36_client_container  | Received acknowledgment: OK
```

Datagramy są poprawnie przesyłane, komunikacja przebiega bez problemu

```
z36_client_container  | Sent datagram of size 32772 bytes, message size: 32768
z36_server_container  | Received datagram of size 32772 from ('172.21.36.3', 39777)
z36_server_container  | Message length validated successfully for message of size 32768 bytes.
z36_client_container  | Received acknowledgment: OK
z36_client_container  | Failed to send datagram of size 65540
z36_client_container  | Failed to send datagram of size 65539
z36_client_container  | Failed to send datagram of size 65538
z36_client_container  | Failed to send datagram of size 65537
```

Przekroczony został maksymalny rozmiar datagramu – do serwera nie dochodzą żadne wiadomości

```
z36_client_container  | Failed to send datagram of size 65508
z36_client_container  | Sent datagram of size 65507 bytes, message size: 65503
z36_server_container  | Received datagram of size 65507 from ('172.21.36.3', 39777)
z36_server_container  | Message length validated successfully for message of size 65503 bytes.
z36_client_container  | Received acknowledgment: OK
z36_client_container  | Maximum datagram size sent: 65507
z36_client_container exited with code 0
```

Maksymalnym rozmiarem datagramu jest **65507 bajtów**. Czyli do teoretycznego maksymalnego rozmiaru brakuje tutaj 28 bajtów. Wynika to z tego, że do datagramu dołączane są nagłówki IP i UDP, które mają odpowiednio 20 i 8 bajtów rozmiaru.