02.12.2024 r.

PSI – Sprawozdanie – Zadanie 2

Autorzy
Krzysztof Gólcz 325159
Daniel Machniak 325190
Natalia Pieczko 325208

1. Treść zadania

Z 2 Komunikacja TCP

Napisz zestaw dwóch programów – klienta i serwera komunikujących się poprzez TCP. Transmitowany strumień danych powinien być stosunkowo duży, nie mniej niż 100 kB.

Zmodyfikuj program klienta tak, aby jednorazowo wysyłane były małe porcje danych (mniejsze od pojedynczej struktury) i wprowadź dodatkowe sztuczne opóźnienie po stronie klienta (przy pomocy funkcji sleep()). W programie serwera zorganizuj kod tak, aby serwer kompletował dane i drukował je „na bieżąco”.

2. Rozwiązanie

Klient

Program klienta został zaimplementowany w Pythonie przy użyciu modułu `socket`. Klient nawiązuje połączenie (`connect`) z serwerem TCP, a następnie przesyła strumień danych o rozmiarze 100 KB w małych porcjach (128 bajtów każda). Po wysłaniu każdej porcji klient wprowadza opóźnienie 0.1 sekundy, symulując warunki bardziej wymagające dla odbiorcy.

Dane do wysłania są generowane w postaci ciągu znaków `X` o długości 102400 bajtów. Pętla iteruje przez cały strumień danych, wysyłając fragmenty o rozmiarze `chunk_size`. Każda wysyłana porcja jest drukowana w konsoli, co ułatwia monitorowanie postępu transmisji.

```
data_to_send = "X" * 102400
chunk_size = 128
total_sent = 0
delay = 0.1

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect((host, port))
    print(f"Connected to {host}:{port}")

    for i in range(0, len(data_to_send), chunk_size):
        chunk = data_to_send[i : i + chunk_size]
        client_socket.sendall(chunk.encode("utf-8"))
        total_sent += len(chunk)
```

Serwer

Serwer został napisany w Pythonie i działa jako aplikacja oczekująca na połączenie klienta za pomocą `socket`. Najpierw przypisywany jest adres i port za pomocą `bind`, a następnie serwer jest w gotowości do akceptacji żądań (`listen`). Akceptuje połączenie od serwera poprzez `accept`. Po nawiązaniu połączenia serwer odbiera dane w porcjach o maksymalnym rozmiarze określonym przez bufor (domyślnie 512 bajtów). Otrzymane dane są natychmiast dodawane do zmiennej `complete_data`, która przechowuje całość strumienia, a następnie wyświetlane na konsoli.

Serwer obsługuje przesył danych w sposób iteracyjny, co pozwala na bieżąco przetwarzać przychodzące porcje. Każda porcja jest drukowana w konsoli, pokazując jej rozmiar i zawartość. Po zakończeniu transmisji serwer podsumowuje całkowitą ilość odebranych danych.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((host, port))
    server_socket.listen()
    print(f"TCP server up and listening on {host}:{port}")

    conn, addr = server_socket.accept()
    with conn:
        print(f"Connection from {addr}")
        complete_data = b""
        while True:
            data = conn.recv(bufsize)
            if not data:
                break
            complete_data += data
            print()
            print("-" * 50)
            print(
                f"Recieved {len(data)} bytes that means \"{data.decode('utf-8')}\""
            )
            print()
            print(f"Total data length: {len(complete_data)} bytes")
            print("-" * 50)
```

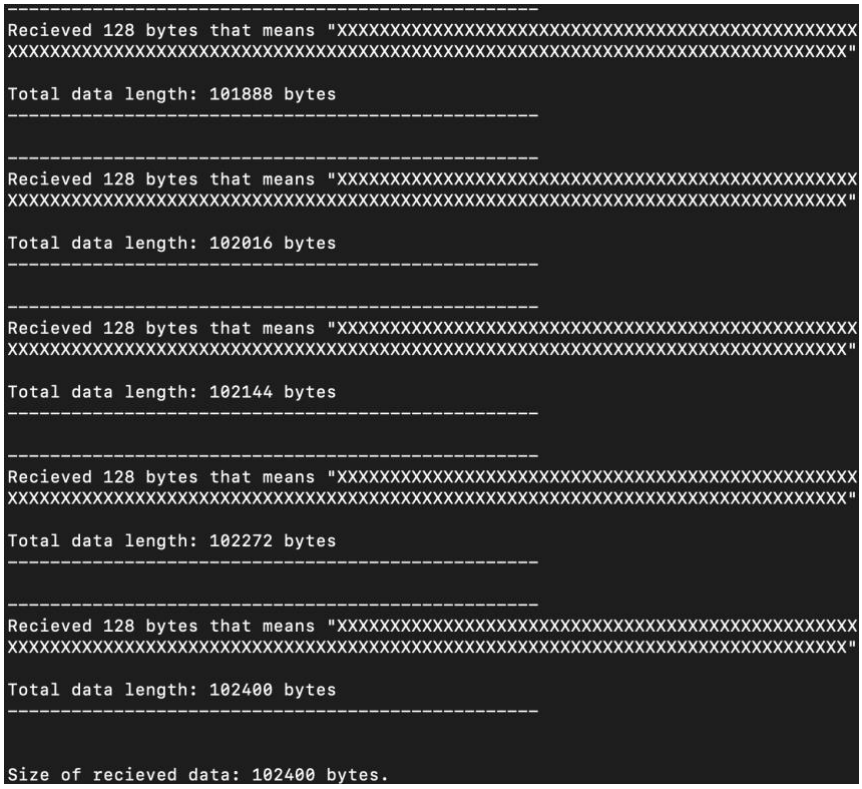
3. Konfiguracja testowa

Serwer

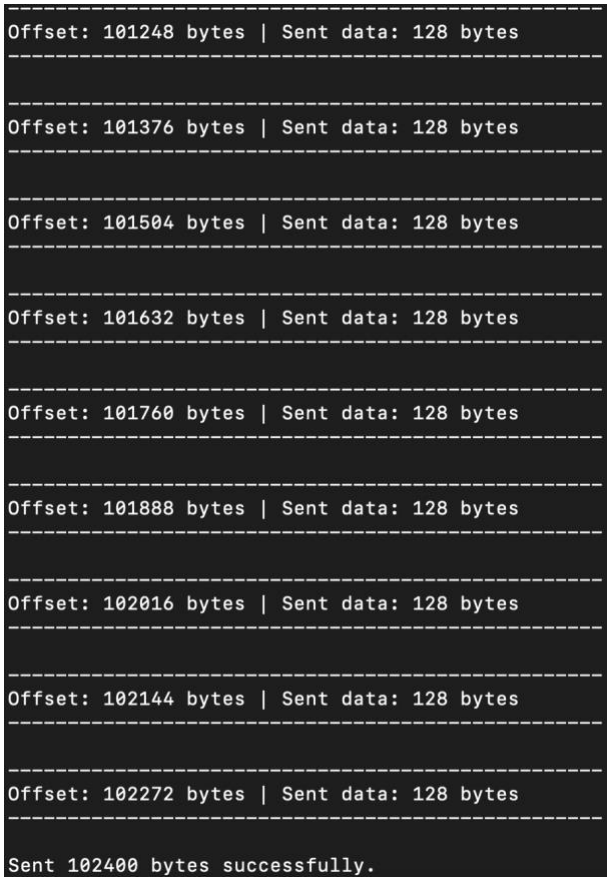
Adres IP: 172.21.33.21
Port: 8080

4. Testowanie

Serwer



Klient



5. Wnioski

Wykorzystanie protokołu TCP zapewniło niezawodność transmisji dużego strumienia danych. Każdy pakiet został dostarczony w odpowiedniej kolejności, co eliminuje potrzebę „manualnego” zarządzania retransmisją i kontrolą poprawności przesyłu.