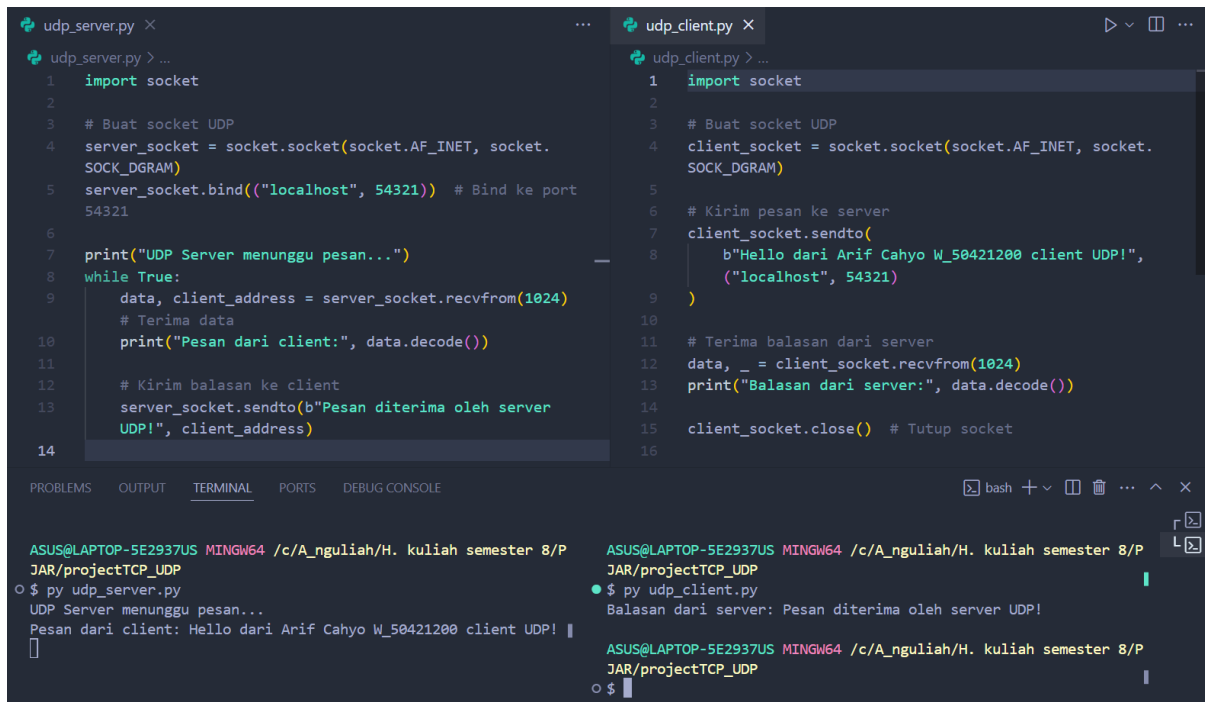


Nama : Arif Cahyo Wibisono
NPM : 50421200
Kelas : 4IA13
Tema : Implementasi UDP & TCP

A. Implementasi UDP (Connectionless)



```
udp_server.py X
1 import socket
2
3 # Buat socket UDP
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5 server_socket.bind(("localhost", 54321)) # Bind ke port 54321
6
7 print("UDP Server menunggu pesan...")
8 while True:
9     data, client_address = server_socket.recvfrom(1024)
10    # Terima data
11    print("Pesan dari client:", data.decode())
12
13    # Kirim balasan ke client
14    server_socket.sendto(b"Pesan diterima oleh server UDP!", client_address)
```

```
udp_client.py X
1 import socket
2
3 # Buat socket UDP
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5
6 # Kirim pesan ke server
7 client_socket.sendto(
8     b"Hello dari Arif Cahyo W_50421200 client UDP!",
9     ("localhost", 54321)
10 )
11
12 # Terima balasan dari server
13 data, _ = client_socket.recvfrom(1024)
14 print("Balasan dari server:", data.decode())
15 client_socket.close() # Tutup socket
```

ASUS@LAPTOP-5E2937US MINGW64 /c/A_nguliah/H. kuliah semester 8/P JAR/projectTCP_UDP
o \$ py udp_server.py
UDP Server menunggu pesan...
Pesan dari client: Hello dari Arif Cahyo W_50421200 client UDP!
o \$

ASUS@LAPTOP-5E2937US MINGW64 /c/A_nguliah/H. kuliah semester 8/P JAR/projectTCP_UDP
JAR/projectTCP_UDP
o \$ py udp_client.py
Balasan dari server: Pesan diterima oleh server UDP!
ASUS@LAPTOP-5E2937US MINGW64 /c/A_nguliah/H. kuliah semester 8/P JAR/projectTCP_UDP
JAR/projectTCP_UDP
o \$

UDP kepanjangannya adalah User Datagram Protocol. Protokol ini digunakan untuk mengirimkan data melalui jaringan komputer dengan cara yang tidak terjamin dan tanpa koneksi. UDP lebih cepat dari protokol TCP karena tidak memerlukan proses jabat tangan untuk membangun koneksi, namun juga kurang andal karena tidak menjamin data sampai dengan benar. Untuk logika programnya seperti dibawah ini:

- a. Udp_server.py
 1. Membuat Socket UDP:
 - socket.AF_INET: Menggunakan alamat IPv4.
 - socket.SOCK_DGRAM: Memilih protokol UDP (berbeda dengan TCP yang menggunakan SOCK_STREAM).
 2. Binding Socket ke Port:
 - bind(): Mengikat socket ke alamat localhost dan port 54321.
 - Port ini bebas dipilih asal tidak digunakan oleh aplikasi lain.
 3. Menerima Pesan dari Client:
 - recvfrom(1024):
 - Menerima data maksimal 1024 byte.
 - Mengembalikan dua nilai:
 - o data: Pesan dari client (dalam bentuk bytes).
 - o client_address: Tuple berisi IP dan port client (misal: ('127.0.0.1', 45678)).

- decode(): Mengubah data bytes ke string (misal: "Hello").
 - 4. Mengirim Balasan ke Client:
 - sendto():
 - o Mengirim pesan balasan (dalam bytes, ditandai b"...") ke alamat client.
 - o client_address digunakan untuk menentukan tujuan pengiriman.
 - 5. Loop Tak Hingga (while True)
 - Server terus berjalan dan siap menerima pesan baru setelah membalas.
- b. Udp_client.py
1. Membuat Socket UDP
 - socket.AF_INET: Menggunakan alamat IPv4.
 - socket.SOCK_DGRAM: Memilih protokol UDP (berbeda dengan TCP yang membutuhkan koneksi terlebih dahulu).
 2. Mengirim Pesan ke Server
 - sendto():
 - o Mengirim data (dalam bentuk bytes, ditandai b"...") ke alamat server.
 - o Parameter:
 - Pesan: b"Hello dari Arif Cahyo W_50421200 client UDP!".
 - Alamat server: ("localhost", 54321) (harus match dengan port server UDP).
 3. Menerima Balasan dari Server
 - recvfrom(1024):
 - o Menerima data maksimal 1024 byte dari server.
 - o Mengembalikan dua nilai:
 - data: Pesan balasan dari server (dalam bytes).
 - _: Alamat server (diabaikan karena sudah diketahui).
 - decode(): Mengubah data bytes ke string untuk dibaca manusia.
 4. Menampilkan Balasan
 - Contoh output: Balasan dari server: Pesan diterima oleh server UDP!
 5. Menutup Socket
 - client_socket.close()
 - Socket ditutup untuk membebaskan sumber daya jaringan.

B. Implementasi TCP (Connection-Oriented)

```
tcp_server.py
1 import socket
2
3 # Buat socket TCP
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 server_socket.bind(("localhost", 12345)) # Bind ke port 12345
6 server_socket.listen(1) # Listen hingga 1 koneksi
7
8 print("TCP Server menunggu koneksi...")
9 connection, client_address = server_socket.accept() # Terima koneksi
10
11 try:
12     print("Terhubung dengan:", client_address)
13     data = connection.recv(1024) # Terima data dari client
14     print("Pesan dari client:", data.decode())
15
16     # Kirim balasan ke client
17     connection.sendall(b"Pesan diterima oleh server TCP!")
18 finally:
19     connection.close() # Tutup koneksi

tcp_client.py
1 import socket
2
3 # Buat socket TCP
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 client_socket.connect(("localhost", 12345)) # Hubungkan ke server
6
7 # Kirim pesan ke server
8 client_socket.sendall(b"Hello Arif Cahyo W_50421200 dari client TCP!")
9
10 # Terima balasan dari server
11 data = client_socket.recv(1024)
12 print("Balasan dari server:", data.decode())
13
14 client_socket.close() # Tutup koneksi

Terminal:
ASUS@LAPTOP-5E2937US MINGW64 /c/A_nguliah/H. kuliah semester 8/P
JAR/projectTCP_UDP
$ py tcp_server.py
TCP Server menunggu koneksi...
Terhubung dengan: ('127.0.0.1', 59327)
Pesan dari client: Hello Arif Cahyo W_50421200 dari client TCP!

ASUS@LAPTOP-5E2937US MINGW64 /c/A_nguliah/H. kuliah semester 8/P
JAR/projectTCP_UDP
$ py tcp_client.py
Balasan dari server: Pesan diterima oleh server TCP!
```

TCP adalah singkatan dari Transmission Control Protocol. Ini adalah salah satu protokol utama dalam suite protokol TCP/IP yang digunakan untuk mentransfer data melalui jaringan komputer. TCP memastikan pengiriman data yang andal dengan mengatur pengiriman paket, mengelola kontrol aliran, dan melakukan pengecekan kesalahan. Untuk logika programnya seperti dibawah ini:

- a. Tcp_server.py
 1. Membuat Socket TCP
 - socket.AF_INET: Menggunakan alamat IPv4.
 - socket.SOCK_STREAM: Memilih protokol TCP (reliable, connection-oriented).
 2. Binding Socket ke Port
 - Mengikat socket ke alamat localhost dan port 12345.
 - Port ini harus unik dan tidak digunakan oleh aplikasi lain.
 3. Listen untuk Koneksi
 - listen(1):
 - o Server siap menerima koneksi.
 - o Angka 1 adalah backlog (antrian maksimum koneksi yang ditahan).
 4. Menerima Koneksi dari Client
 - accept():
 - o Blocking call: Menunggu hingga ada client yang terhubung.
 - o Mengembalikan:
 - connection: Objek socket baru untuk komunikasi dengan client.

- `client_address`: Tuple berisi IP dan port client (misal: ('127.0.0.1', 54321)).
- 5. Menerima dan Mengirim Data
 - `recv(1024)`: Menerima pesan dari client (dalam bentuk bytes, di-decode ke string).
 - `sendall()`: Mengirim balasan ke client (pastikan data dalam bytes, ditandai b"...").
- 6. Menutup Koneksi
 - Koneksi ditutup untuk membebaskan sumber daya, bahkan jika terjadi error.

b. `Tcp_client.py`

1. Membuat Socket TCP
 - `socket.AF_INET`: Menggunakan alamat IPv4.
 - `socket.SOCK_STREAM`: Memilih protokol TCP (koneksi andal berbasis stream).
2. Terhubung ke Server
 - `connect()`:
 - Menghubungkan socket ke alamat server (localhost:12345).
 - Port 12345 harus sama dengan port yang digunakan server TCP.
 - Blocking call: Menunggu hingga koneksi berhasil atau gagal.
3. Mengirim Pesan ke Server
 - `sendall()`:
 - Mengirim data dalam bentuk bytes (ditandai b"...") ke server.
 - Perbedaan dengan `send()`:
 - `sendall()` memastikan semua data terkirim (mengulang pengiriman jika perlu).
 - `send()` mungkin hanya mengirim sebagian data.
4. Menerima Balasan dari Server
 - `recv(1024)`:
 - Menerima data maksimal 1024 byte dari server.
 - Blocking call: Menunggu hingga data diterima atau koneksi terputus.
 - `decode()`: Mengubah data bytes ke string untuk dibaca manusia.
5. Menutup Koneksi
 - Socket ditutup untuk membebaskan sumber daya jaringan.