MODUL 11_ZAFIR ARISTA TOHA_1203210044

Laporan Praktikum : Percobaan Aplikasi Chatting Multicast

Tujuan Percobaan :

Percobaan ini bertujuan untuk mengimplementasikan aplikasi chatting multicast menggunakan protokol UDP untuk berkomunikasi antara pengirim (sender) dan penerima (receiver) pesan. Percobaan ini menggunakan dua receiver untuk menunjukkan fitur multicast, di mana pesan yang dikirim oleh pengirim akan diterima oleh kedua receiver secara bersamaan.

Deskripsi Aplikasi :

Aplikasi ini terdiri dari tiga skrip Python, yaitu sender.py, receiver1.py, dan receiver2.py. Skrip sender.py bertindak sebagai pengirim pesan, sedangkan receiver1.py dan receiver2.py bertindak sebagai dua penerima pesan yang berbeda. Pengirim akan mengirimkan pesan ke alamat multicast yang ditentukan, dan kedua receiver akan menerima pesan tersebut.

Prosedur Percobaan :

1. Jalankan receiver1.py dan receiver2.py di dua terminal atau command prompt yang berbeda.

2. Jalankan sender.py di terminal atau command prompt yang lain.

3. Ketika aplikasi sender.py berjalan, Anda dapat memasukkan pesan yang akan dikirimkan.

4. Perhatikan output pada terminal receiver1.py dan receiver2.py, pesan yang dikirimkan oleh pengirim akan diterima oleh kedua receiver secara bersamaan.

Kode :

sender.py :

```
import socket


group = '224.1.1.1'

port = 5004

# 2-hop restriction in network ttl=2
```

```
ttl = 2

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

while True:
    pesan = input("Masukkan pesan: ")
    sock.sendto(pesan.encode('utf-8'), (group, port))
```

receiver1.py :

```
import socket
import struct

MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5004

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', MCAST_PORT))

mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

while True:
    print(sock.recv(10240).decode('utf-8'))
```

receiver2.py :

```
import socket
import struct
```

```
MCAST_GRP = '224.1.1.1'
MCAST_PORT = 5004

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', MCAST_PORT))

mreq = struct.pack("4sl", socket.inet_aton(MCAST_GRP), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

while True:
    print(sock.recv(10240).decode('utf-8'))
```

Cara menjalankan aplikasi ini adalah sebagai berikut:

1. Jalankan receiver1.py di terminal atau command prompt pertama.
2. Jalankan receiver2.py di terminal atau command prompt kedua.
3. Terakhir, jalankan sender.py di terminal atau command prompt ketiga.

Analisis Hasil Percobaan:

Percobaan ini menunjukkan penggunaan protokol multicast untuk mengirimkan pesan dari satu pengirim ke beberapa penerima secara bersamaan. Hasil percobaan menunjukkan bahwa pesan yang dikirimkan oleh sender.py berhasil diterima oleh receiver1.py dan receiver2.py tanpa perlu pengiriman pesan secara terpisah.

Kesimpulan :

Percobaan aplikasi chatting multicast ini berhasil menunjukkan penggunaan protokol UDP untuk mengirimkan pesan dari pengirim ke beberapa penerima secara bersamaan. Penggunaan multicast memungkinkan pengiriman pesan yang efisien dan skalabilitas yang baik dalam lingkungan jaringan yang sesuai. Namun, dalam implementasi nyata, perlu dipertimbangkan keterbatasan dan aspek keamanan agar aplikasi menjadi lebih efektif dan aman.

Laporan Praktikum: Percobaan Aplikasi Chatting Broadcast (Multithread)

Tujuan Percobaan :

Percobaan ini bertujuan untuk mengimplementasikan aplikasi chatting broadcast (multithread) menggunakan socket di Python. Aplikasi ini memungkinkan beberapa client terhubung ke server dan saling bertukar pesan.

Deskripsi Aplikasi :

Aplikasi ini terdiri dari tiga skrip Python, yaitu server.py, client1.py, dan client2.py. Server.py berfungsi sebagai server yang menerima koneksi dari dua client (client1.py dan client2.py). Setelah koneksi berhasil, server mengirim pesan selamat datang ke masing-masing client. Selanjutnya, client dapat mengirim pesan ke server, dan server akan menyebarkan pesan tersebut ke kedua client yang terhubung.

Prosedur Percobaan :

1. Jalankan server.py di terminal atau command prompt.

2. Jalankan client1.py di terminal atau command prompt untuk menghubungkan ke server.

3. Jalankan client2.py di terminal atau command prompt untuk menghubungkan ke server.

4. Setelah kedua client terkoneksi, mereka dapat saling bertukar pesan dengan server sebagai mediator.

Kode :

server.py :

```
import socket
import threading


def handle_client(client_socket, client_address):
    print(f"Client {client_address} terkoneksi.")
    client_socket.send("Selamat datang di Server...".encode())


    while True:
        message = client_socket.recv(1024).decode()
```

```
        if not message:
            break

        print(f"Pesan dari Client {client_address}: {message}")

        for client in clients:
            if client != client_socket:
                client.send(message.encode())

    client_socket.close()
    print(f"Client {client_address} terputus.")

clients = []

host = socket.gethostname()
port = 8080

server_socket = socket.socket()
server_socket.bind((host, port))
server_socket.listen(2)

print("Proses Dua Koneksi")

while True:
    client_socket, client_address = server_socket.accept()
    clients.append(client_socket)

    client_thread = threading.Thread(target=handle_client, args=(client_socket,
client_address))
    client_thread.start()
```

client1.py :

```
import socket

host = socket.gethostname()
port = 8080

client_socket = socket.socket()
client_socket.connect((host, port))

print("Menyambungkan ke Server")
message = client_socket.recv(1024).decode()
print("Pesan dari Server:", message)

while True:
    message = client_socket.recv(1024).decode()
    print("Server:", message)

    new_message = input("Masukkan Pesan: ")
    client_socket.send(new_message.encode())
    print("Pesan Terkirim")
```

client2.py :

```
import socket

host = socket.gethostname()
port = 8080

client_socket = socket.socket()
client_socket.connect((host, port))
```

```
print("Menyambungkan ke Server")
message = client_socket.recv(1024).decode()
print("Pesan dari Server:", message)

while True:
    message = client_socket.recv(1024).decode()
    print("Server:", message)

    message = client_socket.recv(1024).decode()
    print("Client 1:", message)

    new_message = input("Masukkan Pesan: ")
    client_socket.send(new_message.encode())
    print("Pesan Terkirim")
```

Cara menjalankan aplikasi ini adalah sebagai berikut :

1. Jalankan server.py di terminal atau command prompt.

2. Jalankan client1.py di terminal atau command prompt untuk menghubungkan ke server sebagai client 1.

3. Jalankan client2.py di terminal atau command prompt untuk menghubungkan ke server sebagai client 2.

Analisis Hasil Percobaan :

1. Multithreaded Server: Server menggunakan pendekatan multithreading untuk menangani koneksi dari beberapa client secara bersamaan, mengizinkan server melayani lebih dari satu client tanpa menunggu koneksi selesai.

2. Broadcast Pesan: Setelah client terkoneksi, mereka dapat mengirim pesan ke server, dan server akan menyebarkan pesan tersebut ke semua client yang terhubung, memungkinkan komunikasi broadcast.

3. Keamanan: Tidak ada mekanisme keamanan yang ditambahkan, sehingga pesan yang dikirim oleh satu client dapat dilihat oleh semua client yang terhubung. Dalam implementasi aplikasi nyata, perlu mempertimbangkan lapisan keamanan tambahan agar pesan hanya diterima oleh penerima yang sah.

4. Kelemahan Multithreaded Server: Server memiliki batas pada jumlah koneksi yang dapat ditangani secara bersamaan. Jika jumlah koneksi sangat besar, kinerja server dapat terpengaruh dan mungkin mengalami overhead.

Kesimpulan :

Percobaan aplikasi chatting broadcast (multithread) menggunakan socket di Python berhasil menunjukkan komunikasi antara server dan dua client secara bersamaan. Penggunaan pendekatan multithreading memungkinkan server untuk menangani beberapa koneksi secara efisien. Namun, dalam implementasi nyata, perlu mempertimbangkan keamanan dan skalabilitas agar aplikasi menjadi lebih efektif dan dapat menangani lebih banyak client yang terhubung.

2. Tugas tambahan, proses 4 koneksi dengan tambahkan client menjadi 4 client!

Server :

```
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.bind((host, port))
s.listen(4)  # kita ubah menjadi (4) karena akan terdapat 4 client
print("Waiting for connections...")

clients = []

for i in range(4):
    conn, addr = s.accept()
    clients.append(conn)
    print(f"Client_{i+1} connected")
    conn.send("Selamat datang di Server".encode())

while True:
    message = input("Masukkan Pesan : ")
    message = str(message).encode()
    for client in clients:
        client.send(message)
    print("Pesan Terkirim")
    for client in clients:
```

```
        recv_message = client.recv(1024)  # menerima pesan dari client
        print(f"Client_{clients.index(client) + 1} : ", recv_message.decode())
```

Client 1 :

```
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.connect((host, port))
print("Menyambungkan ke Server")

# Receive the welcome message from the server and decode it from bytes to a
string
message = s.recv(1024)
message = message.decode()
print("Pesan dari server:", message)  # Print the welcome message from the
server

while True:
    # Receive a message from the server and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Server:", message)  # Print the message received from the server

    new_message = input("Masukkan Pesan : ")  # Prompt the client to enter a
new message
    new_message = str(new_message).encode()  # Convert the new message to
bytes (encode it)
    s.send(new_message)  # Send the new message to the server
    print("Pesan Terkirim")  # Print a message indicating that the new message
has been sent

    # Receive a message from Client_2 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_2:", message)  # Print the message received from Client_2

    # Receive a message from Client_3 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_3:", message)  # Print the message received from Client_3

    # Receive a message from Client_4 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
```

```
    print("Client_4:", message)  # Print the message received from Client_4
```

Client 2 :

```
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.connect((host, port))
print("Menyambungkan ke Server")

# Receive the welcome message from the server and decode it from bytes to a
string
message = s.recv(1024)
message = message.decode()
print("Pesan dari server:", message)  # Print the welcome message from the
server

while True:
    # Receive a message from the server and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Server:", message)  # Print the message received from the server

    # Receive a message from Client_1 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_1:", message)  # Print the message received from Client_1

    new_message = input("Masukkan Pesan : ")  # Prompt the client to enter a
new message
    new_message = str(new_message).encode()  # Convert the new message to
bytes (encode it)
    s.send(new_message)  # Send the new message to the server
    print("Pesan Terkirim")  # Print a message indicating that the new message
has been sent

    # Receive a message from Client_3 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_3:", message)  # Print the message received from Client_3

    # Receive a message from Client_4 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_4:", message)  # Print the message received from Client_4
```

Client 3 :

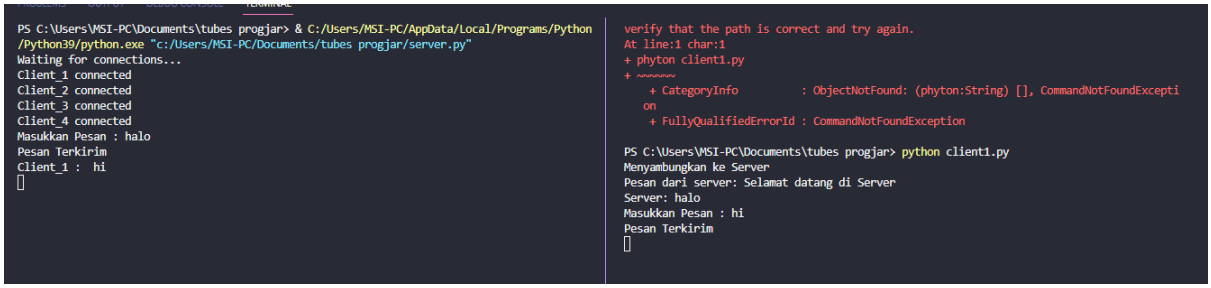
```
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.connect((host, port))
print("Menyambungkan ke Server")

# Receive the welcome message from the server and decode it from bytes to a
string
message = s.recv(1024)
message = message.decode()
print("Pesan dari server:", message)  # Print the welcome message from the
server

while True:
    # Receive a message from the server and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Server:", message)  # Print the message received from the server

    # Receive a message from Client_1 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_1:", message)  # Print the message received from Client_1

    # Receive a message from Client_2 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_2:", message)  # Print the message received from Client_2

    new_message = input("Masukkan Pesan : ")  # Prompt the client to enter a
new message
    new_message = str(new_message).encode()  # Convert the new message to
bytes (encode it)
    s.send(new_message)  # Send the new message to the server
    print("Pesan Terkirim")  # Print a message indicating that the new message
has been sent

    # Receive a message from Client_4 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_4:", message)  # Print the message received from Client_4
```

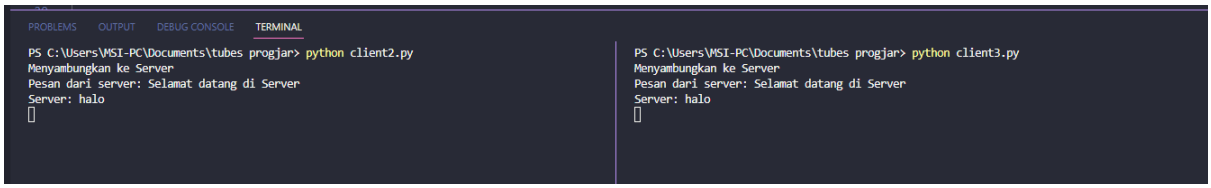
Client 4 :

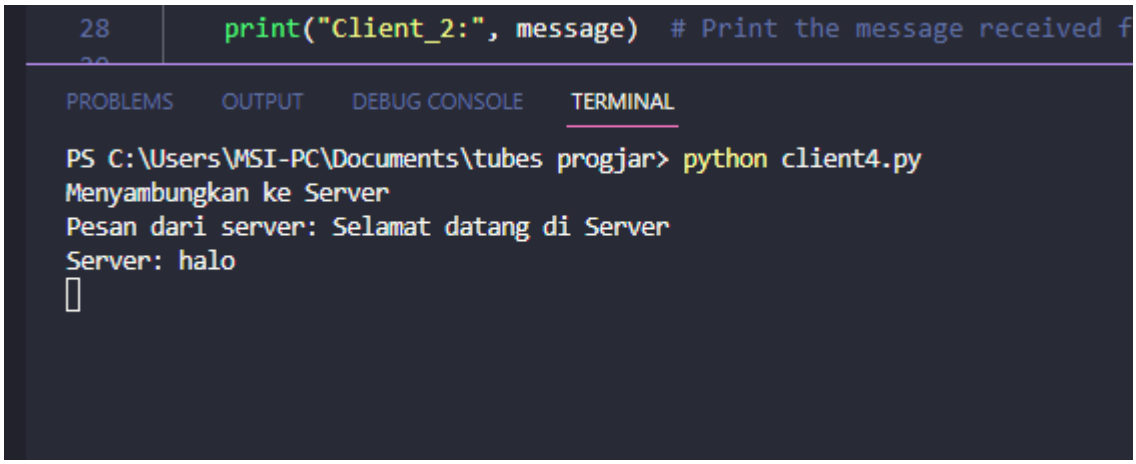
```
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.connect((host, port))
print("Menyambungkan ke Server")

# Receive the welcome message from the server and decode it from bytes to a
string
message = s.recv(1024)
message = message.decode()
print("Pesan dari server:", message)  # Print the welcome message from the
server

while True:
    # Receive a message from the server and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Server:", message)  # Print the message received from the server

    # Receive a message from Client_1 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_1:", message)  # Print the message received from Client_1

    # Receive a message from Client_2 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_2:", message)  # Print the message received from Client_2

    # Receive a message from Client_3 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_3:", message)  # Print the message received from Client_3

    # Receive a message from Client_4 and decode it from bytes to a string
    message = s.recv(1024)
    message = message.decode()
    print("Client_4:", message)  # Print the message received from Client_4

    new_message = input("Masukkan Pesan : ")  # Prompt the client to enter a
new message
    new_message = str(new_message).encode()  # Convert the new message to
bytes (encode it)
    s.send(new_message)  # Send the new message to the server
```

```
    print("Pesan Terkirim")  # Print a message indicating that the new message has been sent
```

Hasil :







00000