**LAPORAN KONSEP JARINGAN**

**[Praktikum 12] Network Programming**



Disusun Oleh:

Aldow Fan Dzikri (NRP. 3121500022)

2 D3 ITA

Dosen Pengampu:

Fitri Setyorini ST. M.Sc

**PROGRAM STUDI D3 TEKNIK INFORMATIKA**

**DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**2022**

Testing Socket

* Kita akan mencoba mendapatkan host name dan ip address dari PC anda
* Buka cmd, ketikkan python

C:\Users\fitri\python

>>> import socket

>>> host\_name = socket.gethostname()

>>> print ("Host name : %s" %host\_name)

Host name : ammar4

>>> ip\_address = socket.gethostbyname(host\_name)

>>> print ("IP address : %s" %ip\_address)

IP address : 192.168.65.14



Cara menjalankan program

* Server
	+ Buka cmd
	+ Jika file udpserver.py di simpan di C:\Users\fitri\python>
	+ Ketikkan
* C:\Users\fitri\python>python udpserver.py
* Client
	+ Buka cmd
	+ Jika file udpclient.py di simpan di C:\Users\fitri\python>
	+ Ketikkan
* C:\Users\fitri\python>python udpclient.py

Udpserver.py

import socket

localIP     = "127.0.0.1"

localPort   = 20001

bufferSize  = 1024

msgFromServer       = "Hello UDP Client"

bytesToSend         = str.encode(msgFromServer)

# Create a datagram socket

UDPServerSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Bind to address and ip

UDPServerSocket.bind((localIP, localPort))

print("UDP server up and listening")

# Listen for incoming datagrams

while(True):

    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

    message = bytesAddressPair[0]

    address = bytesAddressPair[1]

    clientMsg = "Message from Client:{}".format(message)

    clientIP  = "Client IP Address:{}".format(address)

    print(clientMsg)

    print(clientIP)

    # Sending a reply to client

    UDPServerSocket.sendto(bytesToSend, address)

Udpclient.py

import socket

msgFromClient       = "Hello UDP Server ini percobaan pertama"

bytesToSend         = str.encode(msgFromClient)

serverAddressPort   = ("127.0.0.1", 20001)

bufferSize          = 1024

# Create a UDP socket at client side

UDPClientSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

msg = "Message from Server {}".format(msgFromServer[0])

print(msg)

Flow Chart



Penjelasan:

-Pertama udp serve membuat local ip local port dan buffer size yang nantinya akan di bind sebagai media untuk bertukar dengan udp client dimana hal ini dijembatani oleh udp socket datagram

- setelah itu kita membuat msgfromserver untuk pesan yang akan dikirim

-setelah itu apabila berhasil menjalankan bind ke ip,port dan buffer size akan mengeprint server up and listening

-kemudian apabila koneksi berhasil dibuat dengan client akan menerima pesan dari client berupa message dan ip dimana nantinya akan dimasukan pada sebuah variabel dan diprint

-setelah itu server akan mereply dengan pesan yang sudah dibuat sebelumnya

-untuk sisi client hampir sama pertama mengeset ip,port dan buffer size agar sama dengan server setelah itu kita membuat pesan yang mana nantinya akan kita kirimkan ke server, kemudian kita akan mendapat reply dari server berupa pesan

Hasil Running



Cara menjalankan program

* Server
	+ Buka cmd
	+ Jika file tcpserver.py di simpan di C:\Users\fitri\python>
	+ Ketikkan
* C:\Users\fitri\python>python tcpserver.py
* Client
	+ Buka cmd
	+ Jika file tcpclient.py di simpan di C:\Users\fitri\python>
	+ Ketikkan
* C:\Users\fitri\python>python tcpclient.py

Tcpserver.py

import socket

from datetime import datetime

class TCPServer:

    ''' A simple TCP Server for handling a single client '''

    def \_\_init\_\_(self, host, port):

        self.host = host            # Host address

        self.port = port            # Host port

        self.sock = None            # Connection socket

    def printwt(self, msg):

        ''' Print message with current date and time '''

        current\_date\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        print(f'[{current\_date\_time}] {msg}')

    def configure\_server(self):

        ''' Configure the server '''

        # create TCP socket with IPv4 addressing

        self.printwt('Creating socket...')

        self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        self.printwt('Socket created')

        # bind server to the address

        self.printwt(f'Binding server to {self.host}:{self.port}...')

        self.sock.bind((self.host, self.port))

        self.printwt(f'Server binded to {self.host}:{self.port}')

    def wait\_for\_client(self):

        ''' Wait for a client to connect '''

        # start listening for incoming connections

        self.printwt('Listening for incoming connection...')

        self.sock.listen(1)

        # accept a connection

        client\_sock, client\_address = self.sock.accept()

        self.printwt(f'Accepted connection from {client\_address}')

        self.handle\_client(client\_sock, client\_address)

    def get\_phone\_no(self, name):

        ''' Get phone no for a given name '''

        phonebook = {'Alex': '1234567890', 'Bob': '1234512345'}

        if name in phonebook.keys():

            return f"{name}'s phone number is {phonebook[name]}"

        else:

            return f"No records found for {name}"

    def handle\_client(self, client\_sock, client\_address):

        """ Handle the accepted client's requests """

        try:

            data\_enc = client\_sock.recv(1024)

            while data\_enc:

                # client's request

                name = data\_enc.decode()

                resp = self.get\_phone\_no(name)

                self.printwt(f'[ REQUEST from {client\_address} ]')

                print('\n', name, '\n')

                # send response

                self.printwt(f'[ RESPONSE to {client\_address} ]')

                client\_sock.sendall(resp.encode('utf-8'))

                print('\n', resp, '\n')

                # get more data and check if client closed the connection

                data\_enc = client\_sock.recv(1024)

            self.printwt(f'Connection closed by {client\_address}')

        except OSError as err:

            self.printwt(err)

        finally:

            self.printwt(f'Closing client socket for {client\_address}...')

            client\_sock.close()

            self.printwt(f'Client socket closed for {client\_address}')

    def shutdown\_server(self):

        ''' Shutdown the server '''

        self.printwt('Shutting down server...')

        self.sock.close()

def main():

    ''' Create a TCP Server and  respond to client's requests '''

    tcp\_server = TCPServer('127.0.0.1', 4444)

    tcp\_server.configure\_server()

    tcp\_server.wait\_for\_client()

    tcp\_server.shutdown\_server()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

tcpclient.py

import socket

from datetime import datetime

class TCPClient:

    ''' A simple TCP Client that uses IPv4 '''

    def \_\_init\_\_(self, host, port):

        self.host = host        # host address

        self.port = port        # host port

        self.conn\_sock = None   # connection socket

    def printwt(self, msg):

        ''' Print message with current date and time '''

        current\_date\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        print(f'[{current\_date\_time}] {msg}')

    def create\_socket(self):

        ''' Create a socket that uses IPv4 and TCP '''

        self.printwt('Creating connection socket ...')

        self.conn\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        self.printwt('Socket created')

    def interact\_with\_server(self):

        ''' Connect and interact with a TCP Server. '''

        try:

            # connect to server

            self.printwt(f'Connecting to server [{self.host}] on port [{self.port}] ...')

            self.conn\_sock.connect((self.host, self.port))

            # send data

            self.printwt('Sending name to server to get phone number ...')

            name = 'Alex'

            self.conn\_sock.sendall(name.encode('utf-8'))

            self.printwt('[ SENT ]')

            print('\n', name, '\n')

            # receive data

            resp = self.conn\_sock.recv(1024)

            self.printwt('[ RECEIVED ]')

            print('\n', resp.decode(), '\n')

            self.printwt('Interaction completed successfully...')

        except OSError as err:

            self.printwt('Cannot connect to server')

            print(err)

        finally:

            # close socket

            self.printwt('Closing connection socket...')

            self.conn\_sock.close()

            self.printwt('Socket closed')

def main():

    ''' Create a TCP Client and interact with the server at 127.0.0.1:4444'''

    tcp\_client = TCPClient('127.0.0.1', 4444)

    tcp\_client.create\_socket()

    tcp\_client.interact\_with\_server()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Penjelasan:

Untuk progam tcp ini berbeda dengan udp dimana disini menggunakan three way hand shake dimana kita perlu membuka conection terlebih dahulu antara kedu belah pihak berupa connection socket setelah itu baru kita menyambung ke local ip dan local port yang telah dibuat yakni 127.0.0.1 dan 4444 dan harus mengaccept dari sisi server agar client dapat mengirim pesan setelah server dapat mengirim pesan balik dimana disini memerlukan acknowledge yang berbeda dengan udp

Hasil Running:

