**LAPORAN PRAKTIKUM**

**PEMROGRAMAN SOCKET DENGAN PYTHON**

|  |  |
| --- | --- |
| Nama | : L Hafidl Alkhair |
| Nim | : 2023903430060 |
| Kelas | : TRKJ-2C |
| Jurusan | : Teknologi Informasi dan Komputer |
| Progam Studi | : Teknologi Rekayasa Komputer dan Jaringan |
| Dosem Pengampu | : Afla Nevrisa S.Kom, M.Kom |

****

**PROGRAM STUDI TEKNOLOGI REKAYASA KOMPUTER JARINGAN JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER**

**POLITEKNIK NEGERI LHOKSEUMAWE 2025**

# LEMBAR PENGESAHAN

|  |  |
| --- | --- |
| No. Praktikum | : 01 /TIK/TRKJ-2C/ Pemrograman Jaringan |
| Judul | : Laporan Praktikum |
| Nama | : L Hafidl Alkhair |
| Nim | : 202390343060 |
| Kelas | : TRKJ-2C |
| Jurusan | : Teknologi Informasi Dann Komputer |
| Program Studi | : Teknologi Rekayasa Komputer Jaringan |
| Tanggal Praktikum | : 18 Maret 2025 |
| Tanggal Penyerahan | : 15 Maret 2025 |

|  |
| --- |
| Buketrata, 15 Maret 2025 |
| Dosen Pembimbing, |
| **Afla Nevrisa S.Kom, M.Kom**  NIP. 199211172022032007 |

# DAFTAR ISI

[LEMBAR PENGESAHAN i](#_bookmark0)

[DAFTAR ISI ii](#_bookmark1)

[BAB I 1](#_bookmark2)

1. [Tujuan 1](#_bookmark3)
2. [Dasar Teori 1](#_bookmark4)
3. [Alat dan bahan 2](#_bookmark5)

[BAB II 3](#_bookmark6)

1. [Program mengirim pesan [single.py] 3](#_bookmark7)
2. [Program chating dengan Socket internet menggunakan protocol TCP program](#_bookmark8) [Server[pySChat.py] 7](#_bookmark8)
3. [Program Client pyCChat.py 10](#_bookmark9)
4. [Time Server dengan socket Internet menggunakan UDP program time- srv-udp.py](#_bookmark10) [13](#_bookmark10)
5. [Program Time Client [time-cli-udp.py] 15](#_bookmark11)
6. [Program Chatting dengan socket UNIX menggunakan protocol TCP program](#_bookmark12) [Server[pyUnixS.py] 18](#_bookmark12)
7. [Program UnixC.py 21](#_bookmark13)
8. [Program FTP Server dan Client Sederhana Program FTP Server [pyFTPd.py] 23](#_bookmark14)
9. [Program Client [pyFTP.py] 30](#_bookmark15)

[BAB III 36](#_bookmark16)

1. [Kesimpulan 36](#_bookmark17)
2. Tujuan

# BAB I

PENDAHULUAN

* 1. Memahami konsep dasar pemrograman socket dalam jaringan komputer.
  2. Mampu mengimplementasikan komunikasi client-server menggunakan Python.
  3. Mengetahui cara membuat aplikasi sederhana berbasis socket untuk pertukaran data antara client dan server.

1. Dasar Teori

Socket itu seperti pipa penghubung antara dua program yang memungkinkan mereka saling berkirim pesan, baik dalam satu komputer maupun melalui jaringan. Bayangkan seperti dua orang yang sedang berbicara melalui telepon - satu pihak sebagai server (yang menerima panggilan) dan satu lagi sebagai client (yang melakukan panggilan).

Ada dua cara utama program-program ini berkomunikasi:

* 1. Pertama ada TCP, ini seperti percakapan telepon yang teratur. Sebelum mulai bicara, harus ada koneksi dulu. Pesan akan selalu sampai berurutan dan dipastikan tidak ada yang hilang. Ini cocok untuk hal-hal penting seperti kirim file atau chat.
  2. Kedua ada UDP, ini lebih seperti teriakan di lapangan. Tidak perlu sambungan dulu, langsung kirim pesan saja. Lebih cepat tapi pesan bisa hilang atau datang tidak berurutan. Cocok untuk hal-hal seperti streaming video atau game online dimana kecepatan lebih penting daripada akurasi.

Di Python, kita menggunakan modul 'socket' untuk membuat koneksi ini. Server akan membuat socket, menentukan alamatnya, lalu menunggu client menghubungi. Client membuat socket sendiri dan mencoba menghubungi alamat server. Setelah tersambung, mereka bisa saling mengirim data bolak- balik.

Yang menarik, socket bisa digunakan baik untuk komunikasi dalam satu komputer (seperti dua program saling bicara) maupun antar komputer berbeda dalam jaringan. Prinsipnya sama saja, hanya alamatnya yang berbeda - dalam

satu komputer pakai alamat file khusus, sedangkan antar komputer pakai alamat IP dan nomor port.

Pemrograman socket ini menjadi dasar dari semua komunikasi jaringan modern, mulai dari browsing web sampai aplikasi chat. Dengan memahami konsep sederhana ini, kita sudah bisa membuat berbagai macam program jaringan yang berguna.

1. Alat dan bahan
   1. Lapotop os ubuntu/windows
   2. Bahasa python
   3. Vscode

# BAB II

PRAKTIKUM

1. Program mengirim pesan [single.py]
   1. Program

#!/usr/bin/env python import socket

import sys

# Untuk welcome string

welstr = '\n\rSelamat datang di dr\_slump chat server \n\r\ Powered by dr\_slump Technology\n\r\n\r\

User Access Verification\n\r\ Password: '

class Net:

def init (self):

# Cek argumen, jika tidak sama dengan 3 # tampilkan cara penggunaan

if len(sys.argv) != 3:

print("Usage: " + sys.argv[0] + " <hostip> <port>") sys.exit(1)

else:

self.HOST = sys.argv[1] # Set nilai host self.PORT = int(sys.argv[2]) # Set nilai port

self.prompt = 'chat> ' # Prompt yang akan ditampilkan

def Create(self): try:

# Buat socket INET dengan protocol TCP

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) except:

print("Buat socket error...") sys.exit(1)

else:

# Binding ip dan port s.bind((self.HOST, self.PORT)) # Mendengarkan koneksi s.listen(2)

print(f"Server listening on {self.HOST}:{self.PORT}...")

# Menerima koneksi yang datang koneksi, alamat = s.accept()

print(f"Connection established with {alamat}")

# Setelah koneksi diterima, server mengirim pesan selamat datang ke client

koneksi.send(welstr.encode())

stat = 0 # Flag untuk status koneksi # 0=tdk terkoneksi, 1=terkoneksi while True:

# Terima data

data = koneksi.recv(100).decode() if not data:

break

if stat == 0:

if data.strip() == "password": stat = 1

isi = self.prompt

koneksi.send("Anda berhasil login ke server\n\r".encode())

else:

isi = 'Password: '

else:

if data[:8] == 'hostname': host = data.split(' ')[1]

self.prompt = host.strip() + '> ' isi = self.prompt

elif data.strip() in ['keluar']:

koneksi.send("Closing connection...\n".encode()) koneksi.close()

break else:

isi = self.prompt

print(f"Data diterima: {data}") koneksi.send(isi.encode())

s.close()

print("Server closed.")

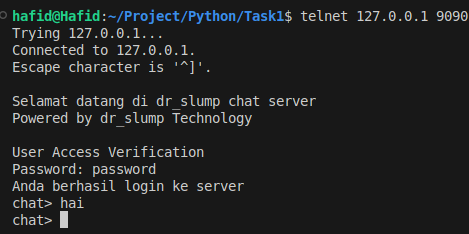
# Inisialisasi dan jalankan server if name == ' main ':

net = Net() net.Create()

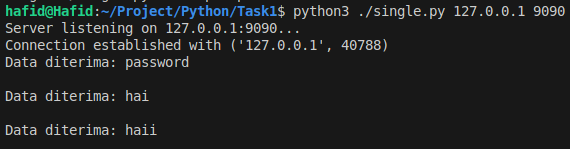
* 1. Penjelasan Kode
     1. Impor Modul:
        + socket digunakan untuk membuat koneksi TCP.
        + sys digunakan untuk mengambil argumen dari command line (IP dan port).
     2. Welcome Message (welstr):
        + Pesan sambutan yang dikirim ke client saat pertama kali terhubung ke server.
     3. Kelas Net:
        + Menyimpan IP dan port dari argumen yang diberikan user saat menjalankan program.
        + self.prompt adalah teks yang muncul sebagai petunjuk input saat user sudah login.
     4. Metode Create():
        + Membuat socket TCP.
        + Melakukan bind ke IP dan port.
        + listen() untuk menunggu koneksi client.
        + Setelah client terkoneksi (accept()), server mengirim welcome message.
     5. Proses Login dan Komunikasi:
        + Status login awal (stat = 0).
        + User diminta memasukkan password.
        + Jika password "password", login berhasil (stat = 1), dan user dapat menggunakan prompt.
     6. Perintah dari Client:
        + hostname <nama>: mengubah prompt.
        + keluar: menutup koneksi.
        + Perintah lain hanya mengembalikan prompt tanpa aksi.
        + Jika client disconnect atau keluar, socket server ditutup (s.close()).
  2. Hasil



Buka terminal baru



Pada server nya dapat



1. Program chating dengan Socket internet menggunakan protocol TCP program Server[pySChat.py]
   1. Program

#!/usr/bin/env python from socket import \* import sys

class TCPServer: def init (self):

if len(sys.argv) != 3:

print('Penggunaan: ' + sys.argv[0] + ' [ip\_address] [nomor\_port]')

sys.exit(1) else:

self.HOST = sys.argv[1] self.PORT = int(sys.argv[2])

def Create(self): try:

# Buat socket TCP

self.sockTCP = socket(AF\_INET, SOCK\_STREAM) # Binding port

self.sockTCP.bind((self.HOST, self.PORT)) # Listening

self.sockTCP.listen(1) except Exception as e:

print(f'Socket error [ip dan port harus valid]: {e}') sys.exit()

else:

print(f'Server Message [tekan Ctrl-C untuk keluar]') print(' ')

print(f'Mendengarkan pada port {self.PORT}')

def Accept(self): try:

koneksi, alamat = self.sockTCP.accept() # Terima koneksi print(f'Koneksi dari {alamat}')

while True:

# Lakukukan perulangan selama true

diterima

mengirim

data = koneksi.recv(1024).decode() # Decode data yang

if not data:

print('Client disconnected.') break # Jika client disconnect

print(f'Pesan dari client >> {data}') # Cetak pesan client

if len(data) > 1:

# Kirim pesan balik

response = f'[{data}] sudah diterima server.' koneksi.send(response.encode()) # Encode sebelum

except Exception as e:

print(f'Error during communication: {e}') finally:

koneksi.close() # Always close the connection after communication ends

def Run(self): self.Create() self.Accept()

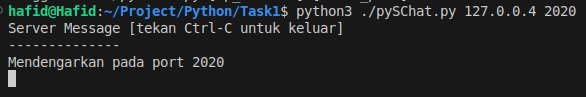
def del (self): try:

self.sockTCP.close() # Tutup koneksi except:

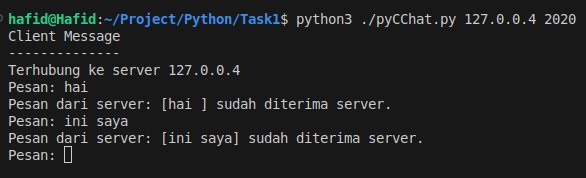
pass

if name == ' main ': server = TCPServer() server.Run()

* 1. Penjelasan kode
     1. Import Modul
        + socket: digunakan untuk membuat koneksi jaringan.
        + sys: digunakan untuk membaca argumen dari command line.
     2. Inisialisasi Kelas ( init )
        + Program mengecek apakah pengguna memberikan dua argumen: IP dan PORT.
        + Jika tidak, akan keluar dengan menampilkan cara penggunaan yang benar.
        + ika benar, nilai IP dan port disimpan dalam variabel self.HOST dan self.PORT.
     3. Metode Create()
        + Membuat socket TCP menggunakan AF\_INET (IPv4) dan SOCK\_STREAM (TCP).
        + Binding socket ke IP dan port yang diberikan.
        + Menjalankan mode listen dengan listen(1) untuk menunggu koneksi dari satu client.
     4. Metode Accept()
        + Menerima koneksi dari client dengan accept().
        + Menampilkan alamat IP client yang terkoneksi.
        + Memasuki loop untuk terus menerima pesan dari client.
        + Jika pesan diterima:
        + Dicetak ke terminal server.
        + Dibalas kembali ke client dengan konfirmasi bahwa pesan sudah diterima.
        + Jika client disconnect, keluar dari loop dan tutup koneksi.
     5. Metode Run()
        + Menjalankan server dengan memanggil Create() lalu Accept().
     6. Metode del ()
        + Dipanggil saat objek dihancurkan (misalnya saat program berhenti).
        + Menutup socket TCP jika masih terbuka.
     7. Bagian Utama (if name == ' main ':)
        + Membuat objek TCPServer dan langsung menjalankan server dengan Run().
  2. Hasilnya



Jika tampilan seperti itu kita tinggal jalankan untuk file pyCChat.py sebagai client



Untuk program dan penjelasan program untuk file pyCChat.py bisa dilihat pada point berikut nya.

1. Program Client pyCChat.py
   1. Program

#!/usr/bin/env python from socket import \* import sys

class TCPClient: def init (self):

if len(sys.argv) != 3:

print('Penggunaan: ' + sys.argv[0] + ' [ip\_server] [port]') sys.exit(1)

else:

self.HOST = sys.argv[1]

self.PORT = int(sys.argv[2]) # Ensure port is an integer

def Create(self): try:

self.sockTCP = socket(AF\_INET, SOCK\_STREAM) # Buat socket TCP

self.sockTCP.connect((self.HOST, self.PORT)) # Connect to the

server

except Exception as e:

print(f'Socket error [ip\_server dan port harus valid]: {e}') sys.exit()

else:

print('Client Message') print(' ')

print(f'Terhubung ke server {self.HOST}') def Send(self):

try:

while True:

pesan = input('Pesan: ') # Input pesan (raw\_input() -> input() in Python 3)

self.sockTCP.send(pesan.encode()) # Send the message (encode to bytes)

server

data = self.sockTCP.recv(1024) # Receive response from the

print(f'Pesan dari server: {data.decode()}') # Decode and

print the response

except KeyboardInterrupt: print("\nConnection closed by user.")

except Exception as e: print(f'Error: {e}')

finally:

self. del () def del (self):

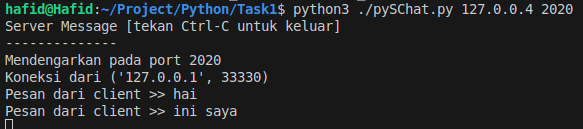
if hasattr(self, 'sockTCP'): self.sockTCP.close() # Close the connection

def Run(self): self.Create() self.Send()

if name == ' main ': client = TCPClient() client.Run()

* 1. Penjelasan kode
     1. Impor Modul
        + socket: digunakan untuk koneksi jaringan.
        + sys: untuk membaca argumen dari terminal.
     2. Konstruktor init ()
        + Mengecek apakah jumlah argumen benar (2 argumen: IP server dan port).
        + Jika benar, menyimpan ke variabel self.HOST dan self.PORT.
     3. Metode Create()
        + Membuat socket TCP dengan socket(AF\_INET, SOCK\_STREAM).
        + Menghubungkan socket ke server menggunakan connect().
        + Jika berhasil, tampilkan pesan bahwa client sudah terhubung.
     4. Metode Send()
        + Melakukan loop agar client dapat terus mengirim pesan.
        + Mengirim pesan ke server setelah input dari pengguna.
        + Menerima balasan dari server dan menampilkannya.
        + Jika terjadi KeyboardInterrupt (Ctrl + C), maka koneksi ditutup dengan aman.
     5. Metode del ()
        + Dipanggil saat objek dihapus.
        + Digunakan untuk menutup socket agar koneksi tidak menggantung.
     6. Metode Run()
        + Memanggil Create() untuk membangun koneksi dan Send() untuk komunikasi.
     7. Program Utama
        + Membuat objek dari kelas TCPClient dan menjalankan Run().
  2. Hasilnya

ini hasilnya dari server



Untuk client nya dapat dilihat pada point diatas

1. Time Server dengan socket Internet menggunakan UDP program time- srv-udp.py
   1. Program

#!/usr/bin/env python import sys

import time

from socket import \*

class ServerWaktu: def init (self):

if len(sys.argv) != 3:

print("Penggunaan: " + sys.argv[0] + " <ip\_address> <port>") sys.exit(1)

self.HOST = sys.argv[1] self.PORT = int(sys.argv[2])

self.tHostPort = (self.HOST, self.PORT)

def buatSocket(self): try:

self.tsock = socket(AF\_INET, SOCK\_DGRAM) print("Mengikatkan IP dengan port " + str(self.tHostPort)) self.tsock.bind(self.tHostPort)

except Exception as e: print("Gagal buat socket: ", e) sys.exit(1)

def tungguKoneksi(self):

print("Menunggu koneksi (Ctrl+C untuk keluar)...") while True:

try:

data, client = self.tsock.recvfrom(1024) # Terima data print(f"Permintaan diterima dari {client}: {data.decode()}") wkt = "Waktu server adalah " + time.ctime(time.time()) self.tsock.sendto(wkt.encode(), client) # Kirim waktu ke

client

except KeyboardInterrupt: print("Server dihentikan.") break

except Exception as e:

print("Kesalahan saat menerima data:", e) def tutupKoneksi(self):

try:

print("Tutup koneksi...") self.tsock.close()

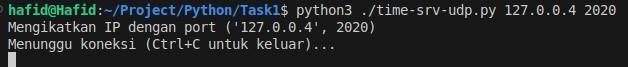
except Exception as e:

print("Gagal menutup koneksi: ", e)

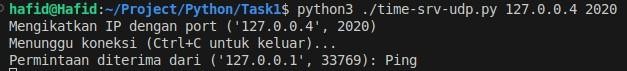
def del (self): self.tutupKoneksi()

if name == " main ": srv = ServerWaktu() srv.buatSocket() srv.tungguKoneksi()

* 1. Penjelasan kode
     1. Import Library
        + sys digunakan untuk mengambil argumen dari command line.
        + time untuk mengambil waktu sistem saat ini.
        + socket untuk membuat koneksi UDP.
     2. Konstruktor init ()
        + Mengecek jumlah argumen yang diberikan user saat menjalankan program.
        + Menyimpan alamat IP dan port ke dalam tuple self.tHostPort.
     3. Fungsi buatSocket()
        + Membuat socket UDP (SOCK\_DGRAM).
        + Mengikatkan socket ke IP dan port yang diberikan (bind()).
        + Menampilkan pesan berhasil atau error jika socket gagal dibuat.
     4. Fungsi tungguKoneksi()
        + Server terus-menerus menunggu permintaan dari client.
        + Jika ada permintaan, server menerima data (recvfrom()).
        + Server mengambil waktu saat ini menggunakan time.ctime() lalu mengirim balik ke client (sendto()).
        + Menangani error jika terjadi kesalahan atau jika pengguna menekan Ctrl+C.
     5. Fungsi tutupKoneksi()
        + Menutup koneksi socket saat server selesai atau dihentikan.
     6. Fungsi del ()
        + Dipanggil otomatis saat objek ServerWaktu dihapus, berguna untuk menutup koneksi secara aman.
     7. Pemanggilan Program
        + Saat file dijalankan langsung, program akan:
          - Membuat objek server.
          - Membuat socket.
          - Menunggukoneksi dari client.
  2. Hasilnya



Ini sedang menunggu koneksi dari client



Jika client sudah konek dengan ip tersebut maka akan tampil seperti ini Yang menandakan ping berarti berhasil.

1. Program Time Client [time-cli-udp.py]
   1. Program

#!/usr/bin/env python import sys

from socket import \*

class ClientWaktu: def init (self):

if len(sys.argv) != 3:

print("Penggunaan: " + sys.argv[0] + " <ip\_address> <port>") sys.exit(1)

self.HOST = sys.argv[1] self.PORT = int(sys.argv[2])

self.tHostPort = (self.HOST, self.PORT) def buatSocket(self):

try:

self.tsock = socket(AF\_INET, SOCK\_DGRAM) print("Menghubungi server waktu di " + str(self.tHostPort)) self.tsock.sendto(b"Ping", self.tHostPort) # Kirim string "Ping" print("Permintaan terkirim ke server")

except Exception as e: print("Gagal buat socket: ", e) sys.exit(1)

def terimaData(self): try:

data, \_ = self.tsock.recvfrom(1024) # Ambil data waktu dari

server

print("Data diterima dari server:", data.decode())

except Exception as e:

print("Gagal menerima data: ", e)

def tutupKoneksi(self): try:

print("Tutup koneksi...") self.tsock.close()

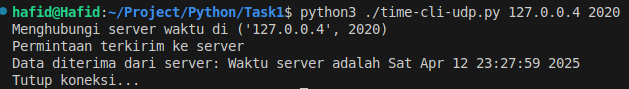
except Exception as e:

print("Gagal menutup koneksi: ", e)

def del (self): self.tutupKoneksi()

if name == " main ": clt = ClientWaktu() clt.buatSocket() clt.terimaData()

* 1. Penjelasan Kode
     1. Library yang Digunakan:
        + sys: Untuk mengambil argumen dari command line (IP dan port).
        + socket: Untuk membuat koneksi UDP (menggunakan SOCK\_DGRAM).
     2. Konstruktor init ()
        + Mengecek apakah argumen yang dimasukkan berjumlah dua (IP dan port).
        + Jika tidak, menampilkan cara penggunaan dan keluar dari program.
        + Menyimpan data IP dan port ke dalam self.tHostPort.
     3. Fungsi buatSocket()
        + Membuat socket UDP.
        + Mengirimkan pesan “Ping” ke server sebagai permintaan waktu.
        + Menampilkan status jika berhasil atau gagal.
     4. Fungsi terimaData()
        + Menunggu dan menerima respon dari server.
        + Menampilkan waktu yang dikirim oleh server setelah didekode dari byte ke string.
     5. Fungsi tutupKoneksi()
        + Menutup socket setelah selesai.
     6. Fungsi del ()
        + Otomatis dipanggil ketika objek dihapus atau program selesai. Fungsinya adalah memastikan koneksi selalu tertutup dengan baik.
     7. Main Program
        + Membuat objek client, mengirim request ke server, lalu menunggu respon waktu dari server.
  2. Hasil



1. Program Chatting dengan socket UNIX menggunakan protocol TCP program Server[pyUnixS.py]
   1. Program

#!/usr/bin/env python import os

import sys

from socket import \*

class TCPServer: def init (self):

self.path = '/tmp/sock' # Path untuk socket UNIX

def create(self): try:

self.sockUnix = socket(AF\_UNIX, SOCK\_STREAM) # Buat socket TCP

self.sockUnix.bind(self.path) # Binding alamat self.sockUnix.listen(1) # Listening untuk koneksi print("Server Message [tekan Ctrl-C untuk keluar]") print(" ")

except OSError:

print(f"Socket sudah digunakan...\n\rJika belum, hapus file

{self.path}")

sys.exit(1)

def accept(self): print("Menunggu koneksi...")

koneksi, alamat = self.sockUnix.accept() # Terima koneksi print("Ada koneksi...")

while True: try:

data = koneksi.recv(1024).decode() # Terima data if not data:

break

print(f"Pesan dari client >> {data}")

koneksi.send(f"[{data}] sudah diterima server.".encode()) #

Kirim respon ke client except:

break

def run(self): self.create() self.accept()

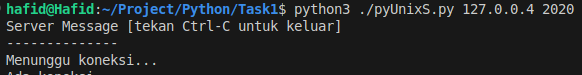
def del (self):

print("Menutup server dan menghapus socket...") self.sockUnix.close()

os.system(f'rm -f {self.path}')

if name == ' main ': server = TCPServer() server.run()

* 1. Penjelasan kode
     1. Library yang Digunakan:
        + os: Untuk menghapus file socket setelah server dihentikan.
        + sys: Untuk menangani error jika socket tidak bisa dibuka.
        + socket: Untuk membuat socket UNIX dengan protokol SOCK\_STREAM.
     2. Konstruktor init ()
        + Menyimpan path file socket UNIX di /tmp/sock. Ini adalah lokasi file socket yang digunakan oleh server untuk mendengarkan koneksi.
     3. Fungsi create()
        + Membuat socket UNIX dengan tipe SOCK\_STREAM (TCP).
        + Melakukan binding alamat file socket pada path yang sudah ditentukan (/tmp/sock).
        + Menunggu koneksi dengan listen(1), yang memungkinkan server untuk menerima 1 koneksi dalam antrian.
        + Jika socket sudah digunakan, program akan menampilkan pesan error dan keluar.
     4. Fungsi accept()
        + Server menunggu koneksi dari client menggunakan accept().
        + Setelah koneksi diterima, server membaca data yang dikirim oleh client menggunakan recv(1024).
        + Jika data ada, server mengirimkan balasan yang berisi pesan dari client, yang menunjukkan bahwa server telah menerima data tersebut.
        + Jika client mengirimkan pesan kosong (disconnect), loop akan berhenti.
     5. Fungsi run()
        + Fungsi utama yang menjalankan server, dengan memanggil create() untuk setup dan accept() untuk menunggu koneksi.
     6. Fungsi del ()
        + Fungsi destructor yang dipanggil saat objek server dihapus.
        + Menutup socket dengan self.sockUnix.close() dan menghapus file socket dengan os.system(f'rm -f {self.path}') untuk memastikan tidak ada file socket yang tertinggal.
     7. Main Program
        + Membuat objek server (server = TCPServer()) dan menjalankan server dengan server.run().
  2. Hasil



Tampilan ini menunjukan server sedang menunggu koneksi dari client

1. Program UnixC.py
   1. Program

#!/usr/bin/env python import sys

from socket import \*

class TCPClient: def init (self):

self.path = '/tmp/sock' # Path untuk koneksi ke socket UNIX

def create(self): try:

self.sockUnix = socket(AF\_UNIX, SOCK\_STREAM) # Buat socket UNIX

self.sockUnix.connect(self.path) # Hubungkan ke server print("Client Message")

print(" ")

except OSError:

print("Socket error [Pastikan server berjalan dan path benar]") sys.exit(1)

def send(self): try:

while True:

pesan = input("Pesan : ") # Meminta input pesan dari

pengguna

server

if not pesan: break

self.sockUnix.send(pesan.encode()) # Mengirim pesan ke

data = self.sockUnix.recv(1024).decode() # Menerima respon

dari server

print(f"Respon dari server: {data}") except KeyboardInterrupt:

print("\nClient dihentikan.") except Exception as e:

print(f"Kesalahan terjadi: {e}") finally:

self. del ()

def run(self): self.create() self.send()

def del (self): print("Menutup koneksi...") self.sockUnix.close()

if name == ' main ': client = TCPClient() client.run()

* 1. Penjelasan kode
     1. Library yang Digunakan:
        + sys: Untuk menangani error jika socket tidak bisa dibuka atau jika terjadi masalah lainnya.
        + ocket: Untuk membuat socket UNIX dengan protokol SOCK\_STREAM yang menggunakan komunikasi TCP.
     2. Konstruktor init ()
        + Menyimpan path file socket UNIX di /tmp/sock. Ini adalah lokasi file socket yang akan digunakan oleh client untuk terhubung dengan server yang sudah berjalan.
     3. Fungsi create()
        + Membuat socket UNIX menggunakan socket(AF\_UNIX, SOCK\_STREAM), yang artinya socket ini menggunakan protokol TCP untuk komunikasi antar proses pada sistem yang sama.
        + Menghubungkan client ke server dengan connect(self.path). self.path adalah lokasi file socket server.
        + Jika terjadi kesalahan pada saat membuat atau menghubungkan socket, program akan menampilkan pesan error dan keluar.
     4. Fungsi send()
        + Program meminta pengguna untuk memasukkan pesan melalui input("Pesan : ").
        + Setelah menerima input, pesan akan dikirimkan ke server menggunakan self.sockUnix.send(pesan.encode()).
        + Client menunggu respon dari server dengan self.sockUnix.recv(1024), dan setelah mendapatkan balasan,

server mengirimkan pesan kembali yang akan dicetak oleh client.

* + - * Jika tidak ada pesan yang dimasukkan oleh pengguna, atau jika pengguna menekan Ctrl+C, maka loop akan berhenti dan koneksi akan ditutup.
    1. Fungsi run()
       - Fungsi utama yang menjalankan client.
       - Pertama, fungsi ini memanggil create() untuk membuat dan menghubungkan socket ke server, lalu melanjutkan untuk mengirim dan menerima pesan dengan send().
    2. Fungsi del ()
       - Fungsi destructor yang akan dipanggil ketika objek client dihapus atau program selesai berjalan.
       - Fungsi ini menutup socket menggunakan self.sockUnix.close() untuk mengakhiri koneksi dengan server.
    3. Main Program
       - Membuat objek client (client = TCPClient()) dan menjalankan client dengan client.run()

1. Program FTP Server dan Client Sederhana Program FTP Server [pyFTPd.py]
   1. Program

#!/usr/bin/env python import os

import sys import time

import subprocess from socket import \*

# Daftar perintah yang didukung lstCmd = [

'dir', 'ls', 'exit', 'bye', 'quit', 'get', 'mget', 'put', 'mput', 'rm', 'delete',

'mv', 'rename', 'cd', 'pwd', 'chmod', 'cp', 'copy', 'rmdir', 'mkdir', 'close', 'disconnect'

]

# Konfigurasi server hostIP = '127.0.0.1'

hostPort = 1111 filFlag = '\*file\*'

getFlag = 'get'

class CMD:

def init (self): self.byeFlag = '\*bye\*'

def checkCmd(self, cmd): ret = ''

cmd = cmd.strip().split() cmd[0] = cmd[0].lower()

if cmd[0] in lstCmd:

if cmd[0] in ['ls', 'dir']:

cmdS = 'ls -al' if len(cmd) == 1 else f'ls {cmd[1]}' ret = subprocess.getoutput(cmdS)

elif cmd[0] in ['rm', 'delete']: if len(cmd) == 2:

ret = subprocess.getoutput(f'rm {cmd[1]}')

ret = f'File {cmd[1]} telah dihapus.' if not ret else ret else:

ret = 'Penggunaan: rm|delete [file].'

elif cmd[0] in ['rmdir']: if len(cmd) == 2:

ret = subprocess.getoutput(f'rm -rf {cmd[1]}')

ret = f'Direktori {cmd[1]} telah dihapus.' if not ret else ret else:

ret = 'Penggunaan: rmdir [direktori].'

elif cmd[0] in ['mkdir']: if len(cmd) == 2:

ret = subprocess.getoutput(f'mkdir {cmd[1]}')

ret = f'Direktori {cmd[1]} telah dibuat.' if not ret else ret else:

ret = 'Penggunaan: mkdir [direktori].'

elif cmd[0] in ['mv', 'rename']: if len(cmd) == 3:

ret = subprocess.getoutput(f'mv {cmd[1]} {cmd[2]}') else:

ret = 'Penggunaan: mv|rename [file\_lama] [file\_baru].'

elif cmd[0] in ['cp', 'copy']: if len(cmd) == 3:

ret = subprocess.getoutput(f'cp {cmd[1]} {cmd[2]}') else:

ret = 'Penggunaan: cp|copy [file\_sumber] [tujuan].'

elif cmd[0] in ['chmod']: if len(cmd) == 3:

ret = subprocess.getoutput(f'chmod {cmd[1]} {cmd[2]}') ret = f'Hak akses {cmd[1]} telah diubah.' if not ret else ret

else:

ret = 'Penggunaan: chmod [mode] [file].'

elif cmd[0] in ['cd']: if len(cmd) == 2:

try:

os.chdir(cmd[1])

ret = f'Direktori sekarang: {os.getcwd()}' except:

ret = 'Direktori tidak ditemukan.'

else:

ret = 'Penggunaan: cd [direktori]'

elif cmd[0] in ['pwd']:

ret = f'Direktori sekarang: {subprocess.getoutput("pwd")}'

elif cmd[0] in ['bye', 'exit', 'quit', 'close', 'disconnect']: ret = self.byeFlag

return ret

class CLI:

def init (self): self.cmd = CMD() self.childLst = []

def updatePid(self, pids): while self.childLst:

pid, status = os.waitpid(0, os.WNOHANG) if not pid:

break pids.remove(pid)

def sendFile(self, sock, file): sock.send(filFlag.encode()) user = os.environ['USER'] size = os.stat(file).st\_size try:

with open(file, 'rb') as f:

sock.send(f"{filFlag}:{file:>214}:{size:>30}:{user:>30}".encode()) while (buffer := f.read(4096)):

sock.send(buffer)

except:

return 0

return 1

def recvFile(self, sock):

msg1 = sock.recv(283).decode().split(':')

flag, namafile, total, user = msg1[0].strip(), msg1[1].strip(), int(msg1[2].strip()), msg1[3].strip()

pjg = 0

if flag == filFlag: try:

with open(namafile, 'wb') as f: while pjg < total:

msg = sock.recv(4096) pjg += len(msg) f.write(msg)

except:

os.remove(namafile) return 0

return 1

return 0

def handler(self, sock):

sock.send("\nSelamat datang di DrSlump FTP Server\n".encode()) while True:

data = sock.recv(1024).decode() if not data:

break

if data.startswith(filFlag): ret = self.recvFile(sock)

sock.send("File telah diterima".encode() if ret else "File gagal diterima".encode())

elif data.startswith(getFlag): cmd = data.strip().split() self.sendFile(sock, cmd[1])

else:

ret = self.cmd.checkCmd(data) if ret == self.cmd.byeFlag:

sock.send("Koneksi ditutup...\n".encode()) sock.close()

break

sock.send((ret + "\n").encode())

def runCmd(self):

print("\n + + + DrSlump FTP Server + + +\n") try:

print("Membuat socket...")

sockCmd = socket(AF\_INET, SOCK\_STREAM) sockCmd.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1) sockCmd.bind((hostIP, hostPort))

sockCmd.listen(5)

print(f"Server berjalan di {hostIP}:{hostPort}") except Exception as e:

print(f"Gagal membuat socket: {e}") sys.exit(1)

while True:

koneksi, alamat = sockCmd.accept()

self.updatePid(self.childLst) pid = os.fork()

if pid:

koneksi.close() self.childLst.append(pid)

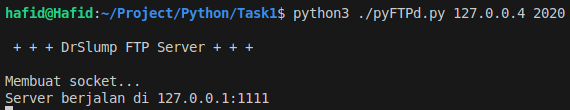
else:

sockCmd.close() self.handler(koneksi) os.\_exit(0)

if name == " main ": cli = CLI()

cli.runCmd()

* 1. Penjelasan kode
     1. Impor dan Daftar Perintah yang Didukung: Program ini mengimpor modul-modul seperti os, sys, time, subprocess, dan socket untuk menangani operasi sistem dan komunikasi melalui socket.
        + Daftar perintah yang didukung (lstCmd) mencakup perintah dasar untuk mengelola file dan direktori, seperti ls, mkdir, rm, get, dan put.
     2. Kelas CMD:
        + Fungsi checkCmd(): Fungsi ini memeriksa perintah yang diterima dari client, mengidentifikasi perintah yang valid dari daftar lstCmd, dan menjalankannya menggunakan sistem perintah melalui subprocess.getoutput(). Misalnya, jika perintah adalah ls, maka perintah sistem ls -al akan dijalankan.
        + Jika perintah terkait pengelolaan file seperti rm, cp, mkdir, dll., maka perintah sistem yang sesuai akan dieksekusi dan outputnya dikembalikan ke client.
          - Beberapa perintah seperti cd dan pwd juga dijalankan langsung dalam program menggunakan pustaka os.
          - Perintah bye, exit, quit, dan lainnya akan menyebabkan server menutup koneksi dan mengirimkan pesan penutupan ke client.
     3. Kelas CLI:
        + Fungsi sendFile(): Fungsi ini menangani pengiriman file dari server ke client. Dimulai dengan mengirimkan header yang berisi informasi file seperti nama file, ukuran file, dan pengguna yang mengirim file. Kemudian, file dibaca dalam potongan kecil (4096 byte) dan dikirimkan ke client.
        + Fungsi recvFile(): Fungsi ini menangani penerimaan file dari client. Setelah menerima header yang berisi informasi file, server membuka file dengan nama yang diterima dan menulis data yang diterima ke dalam file tersebut. Proses ini dilakukan dalam potongan data berukuran 4096 byte.
        + Fungsi handler(): Fungsi ini menangani koneksi dan komunikasi dengan client. Ketika client terhubung, server akan mengirim pesan selamat datang, dan kemudian menunggu perintah dari client.
        + Jika client mengirimkan perintah untuk mengirim atau menerima file (get atau perintah yang dimulai dengan
        + \*file\*), maka fungsi ini akan menangani pengiriman atau penerimaan file.
          - Jika perintah bukan untuk file, server akan memeriksa dan mengeksekusi perintah menggunakan metode checkCmd() yang telah dijelaskan di kelas CMD.
        + Fungsi runCmd(): Fungsi ini adalah titik masuk untuk menjalankan server. Dimulai dengan membuat socket dan mendengarkan koneksi pada alamat IP dan port yang telah ditentukan (127.0.0.1:1111). Ketika koneksi diterima, server membuat proses anak untuk menangani koneksi tersebut menggunakan os.fork().
          - Proses anak akan menangani koneksi, sementara proses induk akan terus menunggu koneksi lainnya.
     4. Main Program:
        + Membuat objek CLI dan menjalankan metode runCmd() yang akan membuat server berjalan dan siap menerima koneksi dari client.
  2. Hasil



Server berhasil dijalankan

I. Program Client [pyFTP.py]

1. Program

#!/usr/bin/env python import os

import sys import string

from socket import \*

# Daftar perintah yang didukung lstCmd = [

'dir', 'ls', 'exit', 'bye', 'quit', 'clear', 'cls', 'get', 'mget',

'put', 'mput', 'rm', 'delete', 'mv', 'rename', 'cd', 'pwd', 'chmod',

'cp', 'copy', '?', 'help', 'rmdir', 'mkdir', '!', 'connect', 'open', 'close', 'disconnect'

]

defPort = 1111 filFlag = '\*file\*'

class IO:

def connect(self, host, port): try:

print('Membuat socket...')

self.sockIO = socket(AF\_INET, SOCK\_STREAM) except:

print('Gagal membuat socket!') return 0

try:

print(f'Koneksi ke {host} port {port}') self.sockIO.connect((host, port))

except:

print('Koneksi gagal!') return 0

print('Koneksi sukses...')

data = self.sockIO.recv(1024).decode() print(data)

return self.sockIO

def sendFile(self, sock, file): sock.send(filFlag.encode())

user = os.environ.get('USER', 'unknown') size = os.stat(file).st\_size

with open(file, 'rb') as f:

buffer = f.read(5000000 - 282)

sock.send(f"{filFlag}:{file:>214}:{str(size).strip():>30}:{user:>30}".e ncode() + buffer)

while (buffer := f.read(5000000)): sock.send(buffer)

def recvFile(self, sock):

msg1 = sock.recv(283).decode().split(':')

flag, namafile, total, user = msg1[0].strip(), msg1[1].strip(), int(msg1[2].strip()), msg1[3].strip()

if flag == filFlag: try:

with open(namafile, 'wb') as f: received = 0

while received < total:

msg = sock.recv(5000000) received += len(msg) f.write(msg)

return 1 except:

os.remove(namafile) return 0

return 0

def close(self): self.sockIO.close()

class CMD:

def init (self): self.getFlag = '\*get\*' self.putFlag = '\*put\*' self.IO = IO() self.isConnected = 0

def checkCmd(self, cmd): cmd0 = cmd.strip() cmd = cmd0.split()

cmd[0] = cmd[0].lower()

if cmd[0] in lstCmd:

if cmd[0] in ['?', 'help']:

print('\nDaftar perintah yang tersedia:\n') print('? help')

print('connect [host] | open [host]') print('ls [direktori] | dir [direktori]') print('cd [direktori]')

print('get [file] | put [file]') print('chmod [mode] [file] | pwd')

print('mkdir [direktori] | rmdir [direktori]') print('rm [file] | delete [file]')

print('mv [file\_lama] [file\_baru] | rename [file\_lama] [file\_baru]')

print('exit | quit | close | disconnect')

elif cmd[0] in ['connect', 'open']: if not self.isConnected:

if len(cmd) == 2:

self.Sock = self.IO.connect(cmd[1], defPort) self.isConnected = bool(self.Sock)

else:

print('Penggunaan: connect|open [host]')

else:

print('Tutup koneksi dulu...')

elif cmd[0] in ['clear', 'cls']: os.system('clear')

elif cmd[0] == 'put': if self.isConnected:

if len(cmd) == 2 and os.path.isfile(cmd[1]): self.IO.sendFile(self.Sock, cmd[1])

else:

print('Gagal membaca file atau format salah.')

else:

print('Penggunaan: put [file]')

elif cmd[0] in ['bye', 'exit', 'quit', 'close', 'disconnect']: if self.isConnected:

self.Sock.send(cmd[0].encode()) print(self.Sock.recv(100).decode()) self.isConnected = 0

else:

print('Goodbye...') sys.exit()

elif cmd[0][0] == '!':

os.system(cmd0[1:])

else:

try:

self.Sock.send(cmd0.encode()) return 1

except:

print('Tidak terkoneksi!') return 0

else:

print('Perintah tidak dikenal.') return 0

def runCmd(self):

print('\n+ + + DrSlump FTP Client + + +\n') while True:

cmd = input('ftp> ').strip() if cmd:

ret = self.checkCmd(cmd)

if self.isConnected and ret == 1:

data = self.Sock.recv(500000).decode() if data.startswith(filFlag):

if self.IO.recvFile(self.Sock) == 0: print('\nFile gagal diunduh!')

else:

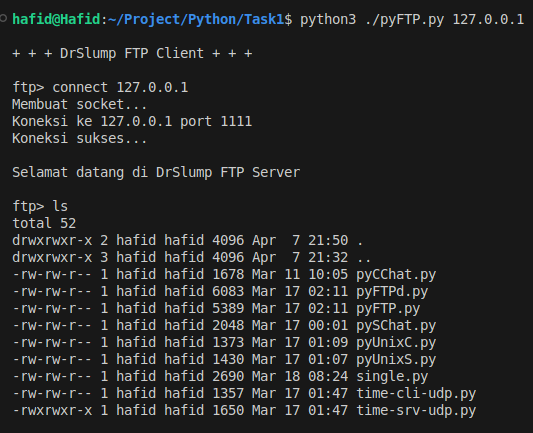
print('\nFile berhasil diunduh...')

else:

print(data)

if name == " main ": cmd = CMD() cmd.runCmd()

1. Penjelasan kode
   1. Script Python ini adalah FTP client sederhana bernama "DrSlump FTP Client", yang bisa digunakan untuk:
      * Terhubung ke server melalui soket TCP.
      * Mengirim dan menerima file (put dan get).
      * Menjalankan perintah mirip Unix/Linux, seperti ls, cd, mkdir, rm, dll.
      * Menampilkan bantuan dengan help atau ?.
      * Menutup koneksi dengan exit, bye, close, dll.
   2. Komponen utama:
      * IO: mengatur koneksi, pengiriman, dan penerimaan file.
      * CMD: menangani parsing perintah dari user dan pengiriman ke server.
      * main: menjalankan antarmuka perintah ftp> secara interaktif. Singkatnya, ini adalah program CLI yang bisa kamu gunakan buat transfer file dan eksekusi perintah ke server yang kompatibel.
2. Hasil



A. Kesimpulan

# BAB III PENUTUP

Kesimpulan dari laporan ini menunjukkan bahwa protokol TCP (Transmission Control Protocol) merupakan protokol komunikasi yang andal karena menyediakan jaminan pengiriman data melalui proses seperti three-way handshake, penggunaan nomor urut, serta mekanisme pengiriman ulang jika terjadi kehilangan data. Praktikum ini berhasil memperlihatkan bagaimana pemodelan client-server dapat dilakukan menggunakan socket TCP dalam bahasa pemrograman Java, sehingga dua komputer dapat saling berkomunikasi melalui jaringan. Server yang dibuat mampu menerima koneksi dari client, memproses pesan yang diterima, serta memberikan respons sesuai instruksi. Selain itu, praktikum ini juga memberikan pemahaman mengenai cara kerja komunikasi blocking yang hanya memungkinkan server untuk menangani satu client dalam satu waktu, serta membuka wawasan tentang perlunya pendekatan multithreading atau asynchronous programming untuk melayani banyak client secara bersamaan.