I. TCP IMPLEMENTATION

1. SERVER

1.1 Import library

Agar kodingan yang dilakukan menjadi lebih mudah maka akan diimpor beberapa library yang berguna untuk melakukan implementasi tcp. Library socket akan digunakan untuk membuat interface TCP. Library os akan digunakan untuk menangani operasi file pada system. Lalu library time akan digunakan untuk menghitung waktu yang telah habis untuk menjalankan proses pengiriman file.

```
import socket
import os
import time
```

1.2 Definisi konstanta

Beberapa konstanta akan digunakan dalam program, maka kita akan kita buat variable dari angka tersebut. Yang pertama ada buffer size, yaitu jumlah byte yang akan dikirimkan secara sekaligus oleh server kepada client. Lalu ada buffer size untuk nama file yaitu buffer filename, hal ini untuk mengatur jumlah byte yang dikirimkan sekaligus saat mengirim nama file. Setelah itu server IP akan di set menjadi localhost yang berarti program hanya akan bekerja pada koneksi local. Dan yang terakhir adalah server port yang akan digunakan.

```
BUFFER_SIZE = 32
BUFFER_FILENAME = 1024
SERVER_IP = 'localhost'
SERVER_PORT = 12345
```

1.3 Socket Setup

Untuk membuat socket server kita akan memanggil library socket, dan melakukan spesifikasi AF_INET untuk menyatakan penggunaan IPv4 dan SOCK_STREAM yang menyatakan penggunaan TCP. Setelah itu kita akan melakukan print statement agar bisa tahu jika server sudah mulai atau belum. Setelah itu kita akan melakukan bind socket pada server ip dan server port yang telah kita definisikan sebelumnya. Lalu kita akan memanggil fungsi listen pada socket, hal ini akan membuat server siap untuk menerima koneksi yang masuk kedalam alamat yang telah kita bind sebelumnya.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('Starting server on port ',SERVER_PORT)

sock.bind((SERVER_IP, SERVER_PORT))

sock.listen(1)
```

1.4 Main Loop

Agar fungsi listen yang kita panggil tetap berjalan maka kita akan membuat sebuah while loop setelahnya. Hal ini untuk memastikan bahwa server tetap berjalan dan tetap menunggu untuk koneksi. Jika server menemukan koneksi maka akan dipanggil fungsi accept, hal ini akan mengambil data dari koneksi tersebut dan memasukkannya kedalam variable.

```
while True:
    print('Waiting for connection')
    connection, client_addr = sock.accept()
```

1.5 Connection Handling

Setelah menerima koneksi kita akan membuat sebuah try statement, hal ini dibuat agar jika ada sebuah eror maka bisa di handle oleh sebuah exception. Dalam statement ini akan dipanggil fungsi time, yaitu untuk memulai perhitungan waktu. Setelah itu dilakukan print statement agar bisa tahu alamat yang terhubung.

Variabel file_name disini digunakan untuk menerima input dari client, yaitu request mengenai file yang mereka inginkan. Setelah menerima nama dari file kita akan membuat file tersebut, dan memulai untuk membaca isi dari file.

```
try:
    start = time.time()
    print(client_addr,' connected')
    file_name = connection.recv(BUFFER_FILENAME).decode()
    file_name = os.path.basename(file_name)

    fd = open(file_name, 'rb')
    buf = fd.read(BUFFER_SIZE)
```

1.6 Sending Data

Dibuat sebuah while loop yang akan berjalan sampai file sudah dibaca dan dikirimkan secara habis. Setelah selesai mengirimkan file, akan ditutup file dan akan diberhentikan fungsi waktu. Setelah itu akan diprint hasil dari proses, yang berisi waktu yang dibutuhkan untuk mengupload dan nama dari file yang telah diupload. Pada di akhir ada sebuah finally statement yang akan berjalan setelah semua process selesai, statement ini akan menutup koneksi yang ada.

```
    while(buf):
        connection.send(buf)
        buf = fd.read(BUFFER_SIZE)

    fd.close()

    end = time.time()
    print(f"Time taken to upload: {end - start} sec")
    print(file_name," uploaded")

finally:
    connection.close()
```

2. CLIENT

2.1 Import library

Agar kodingan yang dilakukan menjadi lebih mudah maka akan diimpor beberapa library yang berguna untuk melakukan implementasi tcp. Library socket akan digunakan untuk membuat interface TCP. Library os akan digunakan untuk menangani operasi file pada system. Lalu library time akan digunakan untuk menghitung waktu yang telah habis untuk menjalankan proses pengiriman file.

```
import socket
import os
import time
```

2.2 Definisi Konstanta

Beberapa konstanta akan digunakan dalam program, maka kita akan kita buat variable dari angka tersebut. Yang pertama ada buffer size, yaitu jumlah byte yang akan dikirimkan secara sekaligus oleh server kepada client. Setelah itu host akan di set menjadi localhost yang berarti program hanya akan bekerja pada koneksi local. Dan yang terakhir adalah server port yang akan digunakan.

```
BUFFER_SIZE = 32
HOST = 'localhost'
PORT = 12345
```

2.3 Input Request

Bagian awal dari program klien dalam komunikasi socket TCP, yang meminta pengguna untuk memilih salah satu dari lima judul buku yang tersedia untuk diunduh dari server. Pengguna diminta memasukkan angka 1 sampai 5, kemudian program mencocokkan input tersebut dengan nama file buku yang sesuai menggunakan if statement. Setelah nama file dipilih, program memisahkan nama dan ekstensi file menggunakan split untuk keperluan pemrosesan nama file lebih lanjut. Terakhir, sebuah socket TCP dibuat, yang digunakan untuk menghubungkan ke server guna memulai proses pengunduhan file.

```
print("Enter the corresponding number to download book:")
print("1. Atlas Shrugged by Ayn Rand")
print("2. Don Quixote by Miguel de Cervantes")
print("3. Shogun by James Clavell")
print("4. The Stand by Stephen King")
print("5. War and Peace by Leo Tolstoy")

file_number = int(input())

if(file_number == 1):
    file_name = "Atlas Shrugged.txt"
elif(file_number == 2):
    file_name = "Don Quixote.txt"
elif(file_number == 3):
    file_name = "Shogun.txt"
elif(file_number == 4):
    file_name = "The Stand.txt"
elif(file_number == 5):
    file_name = "War and Peace.txt"

file = file_name.split('.')

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

2.4 Input Request

Bagian akhir dari blok try-finally, yang memastikan bahwa socket ditutup setelah proses transfer selesai, menggunakan fungsi close. Waktu akhir transfer dicatat dengan end = time.time(), lalu durasi pengunduhan dihitung dan ditampilkan. Selanjutnya, ukuran file hasil unduhan diambil menggunakan os.stat(file_name).st_size. Untuk mengukur performa transfer, throughput dihitung dalam kilobyte per detik dengan membagi ukuran file (dikonversi dari byte ke kilobyte) dengan durasi pengunduhan, lalu dibulatkan ke tiga angka di belakang koma. Akhirnya, nama file yang berhasil diunduh dan nilai throughput ditampilkan ke layar sebagai ringkasan proses.

```
finally:
    sock.close()
    end = time.time()
    print(f"Time taken to download: {end - start} sec")

    file_stat = os.stat(file_name)
    file_size = file_stat.st_size

    throughput = round((file_size*0.001)/(end - start), 3)
    print("Downloaded ",file_name)
    print("Throughput: ",throughput,"kB/s")
```

II. UDP IMPLEMENTATION

1. SERVER

1.1 Inisialisasi

Potongan kode ini merupakan versi server yang menggunakan UDP (bukan TCP) sebagai protokol komunikasinya. Modul socket, os, dan time diimpor, lalu beberapa konstanta ditetapkan, seperti BUFFER_SIZE untuk ukuran potongan data yang dikirim, BUFFER_FILENAME untuk ukuran maksimal nama file, serta alamat dan port server. Perbedaan utama terletak pada pembuatan socket: socket.SOCK_DGRAM digunakan untuk menetapkan bahwa socket ini menggunakan UDP (datagram), bukan SOCK_STREAM seperti pada TCP. Setelah socket dibuat, program mencetak bahwa server dimulai pada port tertentu, lalu mengikat socket ke alamat dan port menggunakan sock.bind(). Karena ini adalah UDP, tidak diperlukan pemanggilan listen() atau accept(), karena koneksi tidak dipertahankan secara terus-menerus.

```
import socket
import os
import time

BUFFER_SIZE = 32
BUFFER_FILENAME = 1024
SERVER_IP = 'localhost'
SERVER_PORT = 12345

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print('Starting server on port ',SERVER_PORT)

sock.bind((SERVER_IP, SERVER_PORT))
```

1.2 Logika Program

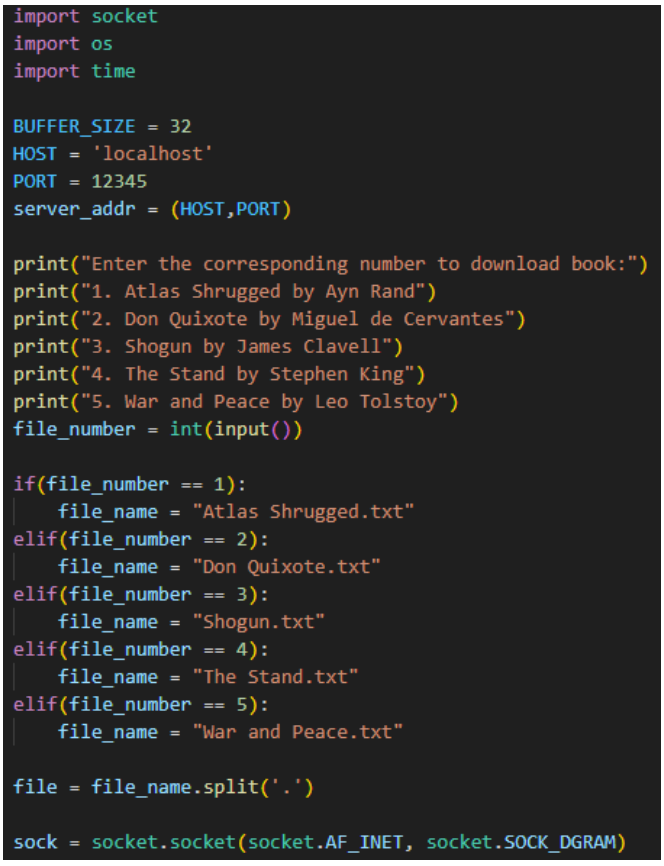
Bagian utama dari server UDP yang terus-menerus menerima permintaan dari klien dalam loop while True. Server menunggu pesan masuk menggunakan sock.recvfrom(BUFFER_FILENAME), yang menerima data beserta alamat pengirim (klien). Setelah nama file diterima dan didekode, nama file difilter menggunakan os.path.basename() untuk keamanan. Server kemudian membuka file dalam mode baca biner ('rb') dan mulai membaca isinya dalam potongan sesuai ukuran BUFFER_SIZE. Data dikirim ke klien menggunakan sock.sendto(buf, client_addr) sampai seluruh file selesai dibaca dan dikirim. Setelah file dikirim sepenuhnya, file ditutup, dan waktu pengiriman dihitung serta dicetak bersama nama file yang berhasil diunggah. Proses ini mencerminkan cara kerja file transfer sederhana melalui protokol UDP, tanpa koneksi tetap.

```
while True:
    print('Waiting to receive message')
    file_name, client_addr = sock.recvfrom(BUFFER_FILENAME)
    print(client_addr,' connected')
    start = time.time()

    file_name = file_name.decode()
    file_name = os.path.basename(file_name)

    fd = open(file_name, 'rb')
    buf = fd.read(BUFFER_SIZE)

    print('Sending Data')

    while(buf):
        sock.sendto(buf, client_addr)
        buf = fd.read(BUFFER_SIZE)

    fd.close()

    end = time.time()
    print(f"Time taken to upload: {end - start} sec")
    print(file_name," uploaded")
```

2. CLIENT

2.1 Inisialisasi

Bagian awal dari program klien yang menggunakan UDP untuk meminta file dari server. Pertama, modul yang diperlukan diimpor, dan beberapa variabel seperti BUFFER_SIZE, alamat server (HOST, PORT), serta server_addr didefinisikan. Selanjutnya, program menampilkan daftar buku dan meminta pengguna untuk memilih salah satu dengan memasukkan nomor. Berdasarkan pilihan tersebut, file_name disesuaikan dengan nama file yang akan diminta. Nama file kemudian dipecah berdasarkan titik (.) untuk memisahkan nama dan ekstensi, kemungkinan untuk diproses lebih lanjut. Socket UDP dibuat dengan socket.SOCK_DGRAM, menandakan bahwa komunikasi ini akan dilakukan tanpa koneksi tetap, berbeda dengan TCP. Bagian ini merupakan tahap persiapan sebelum file diminta dari server.



```
import socket
import os
import time

BUFFER_SIZE = 32
HOST = 'localhost'
PORT = 12345
server_addr = (HOST,PORT)

print("Enter the corresponding number to download book:")
print("1. Atlas Shrugged by Ayn Rand")
print("2. Don Quixote by Miguel de Cervantes")
print("3. Shogun by James Clavell")
print("4. The Stand by Stephen King")
print("5. War and Peace by Leo Tolstoy")
file_number = int(input())

if(file_number == 1):
    file_name = "Atlas Shrugged.txt"
elif(file_number == 2):
    file_name = "Don Quixote.txt"
elif(file_number == 3):
    file_name = "Shogun.txt"
elif(file_number == 4):
    file_name = "The Stand.txt"
elif(file_number == 5):
    file_name = "War and Peace.txt"

file = file_name.split('.')

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

2.2 Pengiriman Data

Bagian utama dari proses pengunduhan file pada sisi klien menggunakan UDP. Setelah mencatat waktu mulai (start = time.time()), klien mengirimkan permintaan nama file ke server menggunakan sock.sendto(...). Nama file yang akan disimpan diubah formatnya untuk mencakup nama file asli, protokol yang digunakan (UDP), PID proses, dan ekstensi file. File kemudian dibuka dalam mode tulis biner ('wb') dan klien mulai menerima data dari server. Setiap iterasi loop menetapkan waktu tunggu (sock.settimeout(2)) selama 2 detik untuk mencegah klien menunggu selamanya jika server berhenti mengirim. Data diterima menggunakan sock.recvfrom(BUFFER_SIZE), dan jika tidak ada data (not byte), proses berhenti. Jika ada data, maka data tersebut ditulis ke file. Ini adalah implementasi sederhana file transfer berbasis UDP yang rentan kehilangan data karena sifat UDP yang tidak andal dan tanpa jaminan urutan atau keutuhan paket.

```
try:
    start = time.time()
    sock.sendto(f"{file_name}".encode(),server_addr)
    file_name = file[0] + "+Protocol=UDP" + "+" + str(os.getpid()) + "." + file[1]

    with open(file_name, 'wb') as f:
        print('Receiving Data')
        while True:
            sock.settimeout(2)

            byte, server = sock.recvfrom(BUFFER_SIZE)

            if not byte:
                break

            f.write(byte)
```

2.3 Penanganan Eror

Bagian penanganan error except yang menangani situasi ketika terjadi timeout saat proses pengunduhan file menggunakan protokol UDP. Jika tidak ada data yang diterima dalam jangka waktu tertentu (2 detik), maka blok ini akan dijalankan. Pertama, pesan "Timeout Occurred" dicetak, lalu waktu akhir dicatat. Perhitungan waktu total pengunduhan dikurangi 2 detik sebagai kompensasi untuk timeout terakhir, kemudian nama file yang berhasil diunduh ditampilkan. Ukuran file dihitung menggunakan os.stat, dan throughput (laju transfer) dihitung dalam kilobyte per detik berdasarkan ukuran file dan waktu yang dihabiskan. Setelah semua informasi ditampilkan, koneksi soket ditutup dua kali secara eksplisit—satu di dalam blok except dan satu lagi di blok finally, meskipun satu penutupan saja sudah cukup.

```
except:
    print("Timeout Occurred")
    end = time.time()
    print(f"Time taken to download: {end - start - 2} sec")
    print("Downloaded ",file_name)

    file_stat = os.stat(file_name)
    file_size = file_stat.st_size

    throughput = round((file_size*0.001)/(end - start), 3)
    print("Throughput: ",throughput,"kB/s")

    sock.close()

finally:
    sock.close()
```