Assignment 02

Name: Darshan Hangoje
Student ID: 202251034

Code:

file_server.py

```
import socket, threading, os

HOST, PORT = "127.0.0.1", 5002

def handle_client(conn, addr):
    print(f"[server] Connected: {addr}")
    try:
        filename = conn.recv(1024).decode().strip()
        if not filename:
            return

        if os.path.exists(filename):
            file_size = os.path.getsize(filename)
            # Send file size + newline as header
            conn.sendall(f"{file_size}\n".encode('utf-8'))
            with open(filename, "rb") as f:
                while chunk := f.read(4096):
                    conn.sendall(chunk)
            print(f"[server] Sent {filename} ({file_size} bytes) to
{addr}")
        else:
            msg = f"File '{filename}' not found\n"
            conn.sendall(msg.encode('utf-8'))
            print(f"[server] {addr} requested missing file: {filename}")
    finally:
        conn.close()
        print(f"[server] Disconnected: {addr}")

def server():
```

```
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((HOST, PORT))
        s.listen()
        print(f"[server] File server running on {HOST}:{PORT}")
        while True:
            conn, addr = s.accept()
            threading.Thread(target=handle_client, args=(conn, addr),
daemon=True).start()

if __name__ == "__main__":
    server()
```

rpc_server.py

```
from xmlrpc.server import SimpleXMLRPCServer
import datetime

HOST, PORT = "127.0.0.1", 6000

def log_request(msg):
    now = datetime.datetime.now().strftime("%H:%M:%S")
    print(f"[server] Request: {msg} at {now}")

def add(x, y):
    result = x + y
    log_request(f"add({x}, {y}) -> {result}")
    return result

def subtract(x, y):
    result = x - y
    log_request(f"subtract({x}, {y}) -> {result}")
    return result

def multiply(x, y):
    result = x * y
    log_request(f"multiply({x}, {y}) -> {result}")
```

```
    return result

def divide(x, y):
    if y == 0:
        log_request(f"divide({x}, {y}) -> Error (division by zero)")
        return "Error: Division by zero"
    result = x / y
    log_request(f"divide({x}, {y}) -> {result}")
    return result

def sum_numbers(numbers):
    result = sum(numbers)
    log_request(f"sum({numbers}) -> {result}")
    return result

def average_numbers(numbers):
    if not numbers:
        return 0
    result = sum(numbers) / len(numbers)
    log_request(f"average({numbers}) -> {result}")
    return result

def max_number(numbers):
    if not numbers:
        return None
    result = max(numbers)
    log_request(f"max({numbers}) -> {result}")
    return result

def main():
    with SimpleXMLRPCServer((HOST, PORT), allow_none=True,
logRequests=False) as server:
        print(f"[server] RPC Calculator running on {HOST}:{PORT}")

        server.register_function(add, "add")
        server.register_function(subtract, "subtract")
        server.register_function(multiply, "multiply")
        server.register_function(divide, "divide")
        server.register_function(sum_numbers, "sum")
        server.register_function(average_numbers, "average")
```

```
        server.register_function(max_number, "max")
        server.serve_forever()

if __name__ == "__main__":
    main()
```

cilent.py

```
import socket
import time
from xmlrpc.client import ServerProxy
import multiprocessing
import os
import math

FILE_SERVER_HOST = '127.0.0.1'
FILE_SERVER_PORT = 5002
RPC_SERVER_HOST = '127.0.0.1'
RPC_SERVER_PORT = 6000
FILENAME = 'number.txt'

def download_file(filename):
    print(f"[client] Requesting file: {filename}")
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((FILE_SERVER_HOST, FILE_SERVER_PORT))
        s.sendall((filename + "\n").encode('utf-8'))
        buf = b''
        while b'\n' not in buf:
            chunk = s.recv(1024)
            if not chunk:
                break
            buf += chunk
        if not buf:
            raise RuntimeError("No response from file server")
        header, rest = buf.split(b'\n', 1)
        try:
            total_size = int(header.decode('utf-8').strip())
        except ValueError:
            msg = (header + rest).decode('utf-8', errors='ignore')
```

```
            raise RuntimeError(f"Server response: {msg}")
        data = rest
        remaining = total_size - len(rest)
        while remaining > 0:
            chunk = s.recv(4096)
            if not chunk:
                break
            data += chunk
            remaining -= len(chunk)
        text = data.decode('utf-8')
        numbers = []
        for line in text.strip().splitlines():
            line = line.strip()
            if not line:
                continue
            try:
                numbers.append(int(line))
            except ValueError:
                pass
        print(f"[client] File received: {len(numbers)} numbers")
        return numbers

def call_rpc(numbers):
    rpc_url = f'http://{RPC_SERVER_HOST}:{RPC_SERVER_PORT}/'
    print("[client] Connecting to RPC Calculator...")
    proxy = ServerProxy(rpc_url, allow_none=True)
    rpc_sum = proxy.sum(numbers)
    rpc_avg = proxy.average(numbers)
    rpc_mx = proxy.max(numbers)
    print(f"RPC sum = {rpc_sum}")
    print(f"RPC average = {rpc_avg}")
    print(f"RPC max = {rpc_mx}")
    return rpc_sum, rpc_avg, rpc_mx

def sequential_sum(numbers):
    start = time.perf_counter()
    s = sum(numbers)
    end = time.perf_counter()
    t = end - start
    return s, t
```

```
def chunked_sum(chunk):
    return sum(chunk)

def parallel_sum(numbers, processes=None):
    if processes is None:
        processes = os.cpu_count() or 2
    n = len(numbers)
    if n == 0:
        return 0, 0.0
    chunk_size = math.ceil(n / processes)
    chunks = [numbers[i:i + chunk_size] for i in range(0, n, chunk_size)]
    start = time.perf_counter()
    with multiprocessing.Pool(processes=len(chunks)) as pool:
        partials = pool.map(chunked_sum, chunks)
    total = sum(partials)
    end = time.perf_counter()
    t = end - start
    return total, t

if __name__ == '__main__':
    try:
        numbers = download_file(FILENAME)
    except Exception as e:
        print("[client] Failed to download file:", e)
        raise SystemExit(1)

    try:
        rpc_sum_value, rpc_avg_value, rpc_max_value = call_rpc(numbers)
    except Exception as e:
        print("[client] RPC call failed:", e)
        raise SystemExit(1)

    seq_sum_value, seq_time = sequential_sum(numbers)
    par_sum_value, par_time = parallel_sum(numbers)

    print(f"Sequential sum = {seq_sum_value}, time = {seq_time:.6f}s")
    print(f"Parallel sum   = {par_sum_value}, time = {par_time:.6f}s")

    if par_time > 0:
```

```
        speedup = seq_time / par_time
        print(f"Speedup: {speedup:.2f}x")
    else:
        print("Parallel time is too small to measure speedup.")
```

Output:

```
PS C:\Users\darsh\Downloads\Lab_2> cd .\Assignment_lab_02\\
PS C:\Users\darsh\Downloads\Lab_2\Assignment_lab_02> python.exe .\file_server.py
[server] File server running on 127.0.0.1:5002
[server] Connected: ('127.0.0.1', 59130)
[server] Sent number.txt (18 bytes) to ('127.0.0.1', 59130)
[server] Disconnected: ('127.0.0.1', 59130)
[server] Connected: ('127.0.0.1', 62874)
[server] Sent number.txt (18 bytes) to ('127.0.0.1', 62874)
[server] Disconnected: ('127.0.0.1', 62874)
```

```
PS C:\Users\darsh\Downloads\Lab_2> cd .\Assignment_lab_02\\
PS C:\Users\darsh\Downloads\Lab_2\Assignment_lab_02> python.exe .\rpc_server.py
[server] RPC Calculator running on 127.0.0.1:6000
[server] Request: sum([10, 20, 30, 40, 50]) -> 150 at 10:45:31
[server] Request: average([10, 20, 30, 40, 50]) -> 30.0 at 10:45:31
[server] Request: max([10, 20, 30, 40, 50]) -> 50 at 10:45:31
[server] Request: sum([10, 20, 30, 40, 50]) -> 150 at 10:46:00
[server] Request: average([10, 20, 30, 40, 50]) -> 30.0 at 10:46:00
[server] Request: max([10, 20, 30, 40, 50]) -> 50 at 10:46:00
```

```
PS C:\Users\darsh\Downloads\Lab_2\Assignment_lab_02> python.exe .\cilent.py
[client] Requesting file: number.txt
[client] File received: 5 numbers
[client] Connecting to RPC Calculator...
RPC sum = 150
RPC average = 30.0
RPC max = 50
Sequential sum = 150, time = 0.000002s
Parallel sum   = 150, time = 0.258422s
Speedup: 0.00x
PS C:\Users\darsh\Downloads\Lab_2\Assignment_lab_02> s
```