COSC264 Assignment 1

Name: Yiyang (Jessie) Yu
Student ID: 48362858

Plagiarism Declaration

This form needs to accompany your COSC 264 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Yiyang Yu

Student ID: 48362858

Signature:

Date: 1/9/2021

server/server.py

```python
"""
Author: Yiyang (Jessie) Yu
Student ID: 48362858
Username: yyu69
Date: 01/08/2021
"""

import datetime
import os
import socket
import sys

SERVER = '0.0.0.0'

MAGIC_NO = 0x497E
HEADER_SIZE = 5

FILEREQUEST_MAX_SIZE = 1024
FILEREQUEST_TYPE = 1

FILERESPONSE_TYPE = 2


def get_port():
    if len(sys.argv) == 2:
        global PORT
        PORT = int(sys.argv[1])
        print(f"[SUCCESS] Port number {PORT} from server side received")
        check_port()
    else:
        print(
```

```
            "[FAIL] Port number from server side missing. \n" +
            "Try run command line again, making sure to add in the end: \n" +
            "- a port number between 1,024 and 64,000 (including)")
        sys.exit(1)


def check_port():
    if PORT < 1024 or PORT > 64000:
        print(
            "[FAIL] Port number must be between 1,024 and 64,000 (including). \n" +
            "Try run command line again, making sure to add in the end: \n" +
            "- a port number between 1,024 and 64,000 (including)")
        sys.exit(1)
    else:
        print(f"[SUCCESS] Port number valid")
        global ADDRESS
        ADDRESS = (SERVER, PORT)


def create_server_socket():
    try:
        global server_socket
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print(f"[SUCCESS] Created server socket: {server_socket}")
    except OSError as err:
        print(f"[FAIL] Trouble creating socket with error: {err}")
        sys.exit(err)


def bind_socket():
    try:
        server_socket.bind(ADDRESS)
        print("[SUCCESS] Socket binded to address (host, port): ", ADDRESS)
    except socket.error:
        print(f"[FAIL] Socket cannot bind to address (host, port): {ADDRESS}")
        sys.exit(1)
```

```python
def start_socket_listening():
    try:
        server_socket.listen()
        print("[SUCCESS] Socket is listening")
    except:
        print("[FAIL] Server socket cannot listen")
        server_socket.close()
        sys.exit(1)


def receive_over_socket(client_socket, size):
    try:
        data = client_socket.recv(size)
        print(f"[SUCCESS] Received {data} of type: {type(data)}")
        return data
    except socket.timeout as error:
        print(f"[FAIL] Trouble receiving the FileRequest. Error: {error}")
        return False


def valid_header():
    if MAGIC_NO != (FILEREQUEST_HEADER[0] << 8) + FILEREQUEST_HEADER[1]:
        print(f"[FAIL] Magic No: {MAGIC_NO} does not match header: {(FILEREQUEST_HEADER[0] << 8) +
FILEREQUEST_HEADER[1]}"
              f"[FAIL] The received FileRequest is erroneous")
        return False

    if FILEREQUEST_TYPE != FILEREQUEST_HEADER[2]:
        print(f"[FAIL] Type: {FILEREQUEST_TYPE} does not match header: {FILEREQUEST_HEADER[2]}"
              f"[FAIL] The received FileRequest is erroneous")
        return False

    global FILENAME_LENGTH
    FILENAME_LENGTH = (FILEREQUEST_HEADER[3] << 8) + FILEREQUEST_HEADER[4]
```

```
    if FILENAME_LENGTH < 1 or FILENAME_LENGTH > 1024:
        print(f"[FAIL] File name length ({FILENAME_LENGTH}) must be between 1 - 1024 including"
              f"[FAIL] The received FileRequest is erroneous")
        return False

    else:
        print(f"[SUCCESS] FileRequest Header pass all condition checks")
        return True


def check_length():
    if len(FILEREQUEST) != FILENAME_LENGTH:

        print(f"[FAIL] File Request length ({len(FILEREQUEST)}) does not make the length specified in the header
({FILENAME_LENGTH})")
    else:
        print(f"[SUCCESS] File Request length ({len(FILEREQUEST)}) matches specifications in the header
({FILENAME_LENGTH})")


def read_file():
    global STATUS_CODE
    global DATA
    try:
        FileRequest_decoded = FILEREQUEST.decode('utf-8')
        print(f"[SUCCESS] FileRequest Decoded: {FileRequest_decoded}")
        DATA = open('server/' + FileRequest_decoded, 'rb')
        print(f"[SUCCESS] Read {DATA} from {FILEREQUEST}")
        STATUS_CODE = 1
        return True
    except OSError as error:
        print(f"[FAIL] {FILEREQUEST} is erroneous. Error: {error}")
        STATUS_CODE = 0
        return False
```

```python
def prepare_FileResponse():
    FileResponse = bytearray()

    FileResponse.append(MAGIC_NO >> 8)
    FileResponse.append(MAGIC_NO & 0x00FF)

    FileResponse.append(FILERESPONSE_TYPE)

    FileResponse.append(STATUS_CODE >> 8)
    FileResponse.append(STATUS_CODE & 0x00FF)

    get_data_length()
    FileResponse.append(DATA_LENGTH >> 24)
    FileResponse.append((DATA_LENGTH  & 0x00FF0000) >> 16)
    FileResponse.append((DATA_LENGTH  & 0x0000FF00) >> 8)
    FileResponse.append(DATA_LENGTH  & 0x000000FF)

    return FileResponse


def get_data_length():
    global DATA_LENGTH
    if STATUS_CODE == 1:
        current_working_directory = os.getcwd()
        path = current_working_directory + '/server/' + FILEREQUEST.decode('utf-8')
        DATA_LENGTH = os.path.getsize(path)
        # DATA_LENGTH = len(DATA).to_bytes(4, byteorder='big')
    if STATUS_CODE == 0:
        DATA_LENGTH = 0x0


def send_over_socket(client_socket, data):
    try:
        client_socket.send(data)
        print(f"[SUCCESS] Sent over socket the data: {data} with type: {type(data)}")
    except socket.error:
```

```
        print(f"[FAIL] Trouble sending data: {data}")
        client_socket.close()
        sys.exit(1)

def main():
    get_port()

    create_server_socket()
    bind_socket()
    start_socket_listening()

    # a forever loop until we interrupt it or
    # an error occurs
    while True:
        # Establish connection with client.
        client_socket, address = server_socket.accept()
        client_socket.settimeout(1)
        print(f"[SUCCESS] Current time: {datetime.datetime.now().time()}")
        print(f"[SUCCESS] Connection from {address} has been establish")

        global FILEREQUEST_HEADER
        FILEREQUEST_HEADER = receive_over_socket(client_socket, HEADER_SIZE)
        if not FILEREQUEST_HEADER:
            client_socket.close()
            continue # go back to start of loop

        if not valid_header():
            client_socket.close()
            continue  # go back to start of loop

        global FILEREQUEST
        FILEREQUEST = receive_over_socket(client_socket, FILEREQUEST_MAX_SIZE)
        if not FILEREQUEST:
            client_socket.close()
            continue # go back to start of loop
```

```
        check_length()

        if not read_file():
            client_socket.close()
            continue

        FileResponse = prepare_FileResponse()
        send_over_socket(client_socket, FileResponse)


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

client/client.py

```python
"""
Author: Yiyang (Jessie) Yu
Student ID: 48362858
Username: yyu69
Date: 01/08/2021
"""

import os
import socket
import sys

HEADER_SIZE = 10
MAGIC_NO = 0x497E
TYPE = 1

FILERESPONSE_HEADER_SIZE = 8
FILERESPONSE_TYPE = 2
FILERESPONSE_SIZE = 4096


def get_command_line_params():
    if len(sys.argv) != 4:
        print(
            f"[FAIL] Incorrect number of parameters \n" +
            "Try run command line again, making sure include in the 3 parameters: \n" +
            "1. An IP address or hostname of computer \n" +
            "2. A port number between 1,024 and 64,000 (including) \n" +
            "3. A file name")
        sys.exit(1)
    else:
        get_IP()
        get_port()
```

```
        get_filename()
        global ADDRESS
        ADDRESS = (IP, PORT)


def get_IP():
    global IP
    try: # Translating a host name to IPv4 address format
        IP = socket.gethostbyname(sys.argv[1]) # Return in dotted-decimal-notation
        print(f"[SUCCESS] IP {IP} from client side received")
    except socket.gaierror:
        print(f"[FAIL] The IP does not exist or not correctly formatted in dotted-decimal-notation")
        sys.exit(1)


def get_port():
    global PORT
    PORT = int(sys.argv[2])
    print(f"[SUCCESS] Port number {PORT} from client side received")
    check_port()


def check_port():
    if PORT < 1024 or PORT > 64000:
        print(
            "[FAIL] Port number must be between 1,024 and 64,000 (including) \n" +
            "Try run command line again, making sure to add as a second parameter: \n" +
            "- a port number between 1,024 and 64,000 (including)")
        sys.exit(1)
    else:
        print(f"[SUCCESS] Port number valid")


def get_filename():
    global FILENAME
    FILENAME = str(sys.argv[3])
```

```python
    print(f"[SUCCESS] Filename {FILENAME} from client side received")
    check_filename()


def check_filename():
    if os.path.isfile(FILENAME):
        print(f"[FAIL] Name of file already exist and can be opened locally")
        sys.exit(1)
    else:
        print(f"[SUCCESS] File name valid")


def create_client_socket():
    try:
        global client_socket
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print(f"[SUCCESS] Created client socket: {client_socket}")
    except OSError as err:
        print(f"[FAIL] Trouble creating socket with error: {err}")
        sys.exit(err)


def connect_socket():
    try:
        client_socket.connect(ADDRESS)
        print(f"[Success] Socket connected to {ADDRESS}")
    except socket.error:
        print(f"[FAIL] Trouble connecting client socket with the server")
        client_socket.close()
        sys.exit(1)


def prepare_FileRequest(filename_bytes):
    filename_length = len(filename_bytes)

    FileRequest = bytearray()
```

```
    FileRequest.append(MAGIC_NO >> 8)
    FileRequest.append(MAGIC_NO & 0x00FF)

    FileRequest.append(TYPE)

    FileRequest.append((filename_length & 0xFF00) >> 8)
    FileRequest.append(filename_length & 0x00FF)

    return FileRequest


def send_over_socket(data):
    try:
        client_socket.send(data)
        print(f"[SUCCESS] Sent over socket the data: {data} with type: {type(data)}")
    except socket.error:
        print(f"[FAIL] Trouble sending data: {data}")
        client_socket.close()
        sys.exit(1)


def receive_over_socket(size):
    try:
        data = client_socket.recv(size)
        print(f"[SUCCESS] Received {data} of type: {type(data)}")
        return data
    except socket.timeout as error:
        print(f"[FAIL] Trouble receiving the FileResponse. Error: {error}")
        return False
    except socket.error as error:
        print(f"[FAIL] Trouble receiving the FileResponse. Error: {error}")
        return False

def valid_header():
    if MAGIC_NO != (FILERESPONSE_HEADER[0] << 8) + FILERESPONSE_HEADER[1]:
```

```
        print(f"[FAIL] Magic No: {MAGIC_NO} does not match header: {(FILERESPONSE_HEADER[0] << 8) +
FILERESPONSE_HEADER[1]}"
              f"[FAIL] The received FileRequest is erroneous")
        client_socket.close()
        sys.exit(1)

    if FILERESPONSE_TYPE != FILERESPONSE_HEADER[2]:
        print(f"[FAIL] Type: {FILERESPONSE_TYPE} does not match header: {FILERESPONSE_HEADER[2]}"
              f"[FAIL] The received FileRequest is erroneous")
        client_socket.close()
        sys.exit(1)

    global STATUS_CODE
    STATUS_CODE = (FILERESPONSE_HEADER[3] << 8) + FILERESPONSE_HEADER[4]
    if STATUS_CODE not in (1, 0):
        print(f"[FAIL] Status code: {STATUS_CODE} is not 1 or 0\n"
              f"[FAIL] The received FileRequest is erroneous")
        client_socket.close()
        sys.exit(1)

    else:
        print(f"[SUCCESS] FileResponse Header pass all condition checks")
        return True

def open_file():
    try:
        file = open(FILENAME, 'wb+')
        print(f"[SUCCESS] Opened file ({file}) locally for writing: {FILENAME}")
        return file
    except:
        print(f"[ERROR] Trouble opening requested filename: {FILENAME}")
        client_socket.close()
        sys.exit(1)
        return False
```

```python
def writing(file, block):
    try:
        file.write(block)
        print(f"[SUCCESS] Written into file ({file}) the block: {block}")
        return True
    except:
        print(f"[FAIL] Trouble writing into file ({file}) the block: {block}")
        return False


def main():

    get_command_line_params()

    create_client_socket()
    connect_socket()

    filename_bytes = FILENAME.encode('utf-8')

    FileRequest = prepare_FileRequest(filename_bytes)
    send_over_socket(FileRequest)

    send_over_socket(filename_bytes)

    global FILERESPONSE_HEADER
    FILERESPONSE_HEADER = receive_over_socket(FILERESPONSE_HEADER_SIZE)

    if valid_header() and STATUS_CODE == 1:
        file = open_file()

    if STATUS_CODE == 0:
        print(f"[RESULT] No File Data is following. Server side does not contain the requested file {FILENAME}")

    byte_count = 0
    processing_file = True
    while processing_file:
```

```
        block = receive_over_socket(FILERESPONSE_SIZE)
        written = writing(file, block)
        print(f"[DEBUGGING] Written is: {written}, block is: {block}")
        if not block or not written:
            client_socket.close()
            file.close()
            sys.exit(1)
            processing_file = False
        byte_count += len(block)
        print(f"[SUCCESS] Byte count is: {byte_count} for block: {block}")

    data_length = (FILERESPONSE_HEADER[4] << 24) | (FILERESPONSE_HEADER[5] << 16) | (
                FILERESPONSE_HEADER[6] << 8) | FILERESPONSE_HEADER[7]

    if byte_count == data_length:
        print(f"[SUCCESS] Number of bytes received ({byte_count}) match with the DataLength indicated in
FileResponse header: {data_length}")
    else:
        print(f"[FAIL] Number of bytes received ({byte_count}) does not match with the DataLength indicated in
FileResponse header: {data_length}")

    client_socket.close()
    file.close()
    sys.exit(1)


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