1. [45 Point]

server_host.py > ...

```python
#import socket module
from socket import *
import time, threading
import sys # In order to terminate the program
serverSocket = socket(AF_INET, SOCK_STREAM)
#Prepare a sever socket
#Fill in start
serverSocket.bind(('', 6789))
serverSocket.listen(1)
#Fill in end
while True:

        #Establish the connection
        print('Ready to serve...')
        #Fill in start

        #Fill in end
        try:
                connectionSocket, addr = serverSocket.accept()
                message = connectionSocket.recv(1024) #Fill in start #Fill in end
                print((message.split()[1])[1:])
                filename = message.split()[1][1:]
                f = open(filename,"r")
                outputdata = f.read() #Fill in start #Fill in end
                f.close()
                #Send one HTTP header line into socket
                #Fill in start
                content= ("HTTP/1.1 200 OK\r\n" +  \
                                                "Content-Length: %d\r\n" % len(outputdata) +  \
                                                "Date: %s \r\n" %  time.strftime('%Y-%m-%d', time.localtime(time.time() ) )  +  \
                                                "Content-Type: text/html;charset=utf-8\r\n\r\n" +  \
                                                outputdata)
                print(content)
                connectionSocket.send(content.encode())

                #Fill in end
                #Send the content of the requested file to the client
                connectionSocket.close()
        except IOError:
                #Send response message for file not found
                #Fill in start
                connectionSocket.send('\nHTTP/1.1 404 Not Found\n\n'.encode())
                #Fill in end
                #Close client socket
                #Fill in start
                connectionSocket.close()

serverSocket.close()
sys.exit()#Terminate the program after sending the corresponding data
```

2. [25 point) dengan threading

```
#import socket module
from socket import *
import time, threading
import sys # In order to terminate the program
serverSocket = socket(AF_INET, SOCK_STREAM)
#Prepare a sever socket
#Fill in start
serverSocket.bind(('', 6789))
serverSocket.listen(5)
#Fill in end
while True:

        #Establish the connection
        print('Ready to serve...')
        #Fill in start

        #Fill in end
        try:
                connectionSocket, addr = serverSocket.accept()
                message = connectionSocket.recv(1024) #Fill in start #Fill in end
                print((message.split()[1])[1:])
                filename = message.split()[1][1:]
                f = open(filename,"r")
                outputdata = f.read() #Fill in start #Fill in end
                f.close()
                #Send one HTTP header line into socket
                #Fill in start
                content= ("HTTP/1.1 200 OK\r\n" +  \
                                        "Content-Length: %d\r\n" % len(outputdata) +  \
                                        "Date: %s \r\n" %  time.strftime('%Y-%m-%d', time.localtime(time.time() ) )  +  \
                                        "Content-Type: text/html;charset=utf-8\r\n\r\n" +  \
                                        outputdata)
                print(content)
                connectionSocket.send(content.encode())

                #Fill in end
                #Send the content of the requested file to the client
                connectionSocket.close()
        except IOError:
                #Send response message for file not found
                #Fill in start
                connectionSocket.send('\nHTTP/1.1 404 Not Found\n\n'.encode())
                #Fill in end
                #Close client socket
                #Fill in start
                connectionSocket.close()

serverSocket.close()
sys.exit()#Terminate the program after sending the corresponding data
```

Client:

```python
from socket import *
import sys
import logging

clientsocket = socket(AF_INET, SOCK_STREAM)

if len(sys.argv) != 4:
    print(len(sys.argv))
    print("Your command is not right. Please be in this format:client.py server_host server_port filename")
    sys.exit(0)

host = str(sys.argv[1])
port = int(sys.argv[2])
request = str(sys.argv[3])
request = "GET /" + request + " HTTP/1.1"
try:
    clientsocket.connect((host,port))
except Exception:
    # print exception cause
    logging.exception("An exception was thrown!")
    print ("Please try again.\r\n")
    sys.exit(0)
clientsocket.send(request.encode())

response = clientsocket.recv(1024)
print(response.decode())
clientsocket.close()
```

3. [10 point] Jalankan server dengan cara run “server.py”, kemudian “client.py localhost 6789 HelloWorld.html”
4. [10 point] Contoh beberapa framework:

Unicorn: Unicorn adalah server HTTP pre-forked dengan model kerja Worker yang memungkinkan kinerja tinggi. Fiturnya termasuk dukungan untuk protokol HTTP dan WSGI (Web Server Gateway Interface).

Contoh sederhana penggunaan Unicorn:

```
def app(environ, start_response):
    data = b"Hello, Unicorn!"
    start_response("200 OK", [
        ("Content-Type", "text/plain"),
        ("Content-Length", str(len(data)))
    ])
   return iter([data])

if __name__ == "__main__":
    from gunicorn.app.wsgiapp import WSGIApplication
```

```
    WSGIApplication("%(prog)s [OPTIONS]").run()
```

Gunicorn: Gunicorn adalah server HTTP untuk Python WSGI aplikasi. Salah satu fitur utamanya adalah kemampuan untuk mengelola banyak koneksi secara efisien.
Contoh sederhana penggunaan Gunicorn:

bash
gunicorn -w 4 myapp:app

CherryPy: CherryPy adalah framework web yang sederhana dan ringan, dengan fokus pada kesederhanaan dan kemudahan penggunaan.
Contoh sederhana penggunaan CherryPy:

```
import cherrypy

class HelloWorld:
    @cherrypy.expose
    def index(self):
        return "Hello, world!"

if __name__ == '__main__':
    cherrypy.quickstart(HelloWorld())
```

Daphne: Daphne adalah server HTTP untuk protokol ASGI (Asynchronous Server Gateway Interface), sering digunakan dengan Django Channels.

Python Trio: Ini bukan sebuah framework, melainkan sebuah pustaka untuk pemrograman konkurensi. Trio menawarkan alat untuk menulis kode yang konkuren dan aman.

Twisted Web: Twisted adalah kerangka kerja asinkron yang menyediakan alat untuk membangun aplikasi jaringan dan web yang skalabel.

AIOHTTP: AIOHTTP adalah kerangka kerja web asinkron yang menggunakan async/await di Python. Ini memungkinkan pemrograman asinkron yang efisien.

Tornado: Tornado adalah kerangka kerja jaringan dan web yang dirancang untuk menangani banyak koneksi bersamaan. Ia juga menyediakan alat untuk pengembangan aplikasi real-time.

Meinheld: Meinheld adalah server WSGI yang cepat untuk aplikasi Python.

Flask: Flask adalah kerangka kerja web ringan yang menyediakan alat untuk membangun aplikasi web dengan mudah, tanpa lapisan tambahan yang kompleks.

5. Code di script 5 lebih rapih dan sudah menerapkan REST API Protocol (terbukti dari penggunaan GET dan HEAD)