COSC 264 Assignment
File Transfer Socket Program

3/8/21

Alexander Robert Burling
88866582
arb142

```
"""
Client script for socket based file downloader
Usage: python3 client.py server_ip server_port target_file

3/8/21
Alex Burling
88866582
"""


import os.path
import socket
import sys
from FileRequest import FileRequest
from FileResponse import FileResponse


#Parses command line arguments and checks validity
def parse_args():

    if len(sys.argv) != 4:
        sys.exit("Usage: python3 client.py server_ip server_port target_file.")

    server_ip = str(sys.argv[1])
    server_port = int(sys.argv[2])
    target_file = str(sys.argv[3])

    if server_port < 1024 or server_port > 64000:
        sys.exit("PORT NUMBER SHOULD BE >= 1,024 AND <= 64,000.")

    try:
        server_addr = socket.gethostbyname(server_ip)
    except socket.gaierror:
        sys.exit("MALFORMED IP/HOSTNAME")

    if (os.path.exists(target_file)):
        sys.exit("The requested file already exists locally.")

    return server_ip, server_port, target_file, server_addr


""" Starts socket and attempts to connect to the provided
    server address and port, exits on connection error"""
def start_socket(server_addr, server_port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        sock.connect((server_addr, server_port))
    except ConnectionRefusedError:
        sock.close()
        sys.exit("{}:{} REFUSED CONNECTION.".format(
            server_addr, str(server_port)))

    return sock


""" Recieves FileResponse from provided socket and counts
    number of bytes recieved. Closes socket once full response
    recieved"""
def recieve_response(sock):
    recieved = bytearray(sock.recv(1024))
    packet_len = len(recieved)
    file_response = FileResponse.from_bytearray(recieved)
    while (not file_response.check_len()):
        recieved = bytearray(sock.recv(4096))
        packet_len += len(recieved)
        file_response.append_data(recieved)
    sock.close()
    return file_response, packet_len


""" Generates a FileRequest from provided filename and
    sends to provided socket"""
def send_request(sock, filename):
    req = FileRequest.from_filename(filename)
    sock.send(req.generate_packet())


""" Writes retrieved data into target file"""
def write_file(file_response, target_file):
    file = open(target_file, 'wb+')
    file.write(file_response.get_data())
    file.close()


""" Main function"""
def run_client():
    server_ip, server_port, target_file, server_addr = parse_args()

    print("REQUESTING '{}' FROM [{}]{}:{}".format(
        target_file, server_ip, server_addr, str(server_port)))

    try:
        sock = start_socket(server_addr, server_port)
        sock.settimeout(1)

        send_request(sock, target_file)

        file_response, bytes = recieve_response(sock)
        print("RESPONSE RECIEVED, RECIEVED {} BYTES".format(bytes))

        file_response.validate()

        write_file(file_response, target_file)
        print("FILE SUCCESSFULLY DOWNLOADED")

    except socket.timeout:
        sock.close()
        print("CONNECTION WITH [{}]{}:{} TIMED OUT".format(
            server_ip, server_addr, str(server_port)))


def main():
    run_client()


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

server.py

```
"""
Server script for socket based file downloader
Usage: python3 server.py listen_port

3/8/21
Alex Burling
88866582
"""



import sys
import socket
from datetime import datetime
from FileRequest import FileRequest
from FileResponse import FileResponse


#Parses command line arguments and checks validity
def parse_args():
    if len(sys.argv) != 2:
        sys.exit("Usage: python3 server.py listen_port")

    port = int(sys.argv[1])
    if port < 1024 or port > 64000:
        sys.exit("PORT NUMBER SHOULD BE >= 1,024 AND <= 64,000")

    return port


""" Starts socket and attempts to bind to the provided
    port, exits on error"""
def start_socket(port):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        hostname = socket.gethostname()
        sock.bind((hostname, port))
        sock.listen()
    except:
        sock.close()
        sys.exit("ERROR OPENING SOCKET")

    return sock, hostname


""" Recieves FileRequest from provided socket"""
def recieve_request(sock):
    recieved = bytearray(sock.recv(1024))
    file_request = FileRequest.from_bytearray(recieved)
    while (not file_request.check_len()):
        recieved = bytearray(sock.recv(1024))
        file_request.append_data(recieved)

    return file_request


""" Generates a FileResponse from provided filename and
    sends to provided socket"""
def send_response(sock, filename):
    res = FileResponse.from_filename(filename)
    packet, packet_len = res.generate_packet()
    client_addr = sock.getpeername()
    print("[{}]  {}:{} SENT {} BYTES".format(datetime.now().strftime(
        "%H:%M:%S"), client_addr[0],  client_addr[1], packet_len))
    sock.send(packet)


""" Main function"""
def run_server():
    port = parse_args()
    sock, hostname = start_socket(port)
    print("Listening on [{}]{}:{}".format(hostname, sock.getsockname()[0], str(port)))

    while True:

        try:
            client_sock, client_address = sock.accept()
            client_sock.settimeout(1)

            print("[{}]  {}:{} CONNECTED".format(datetime.now().strftime(
                "%H:%M:%S"), client_address[0], client_address[1]))

            file_request = recieve_request(client_sock)

            file_request.validate()
            filename = file_request.get_filename()

            print("[{}]  {}:{} GET FILE '{}'".format(datetime.now().strftime(
                "%H:%M:%S"), client_address[0], client_address[1], filename))

            send_response(client_sock, filename)
            client_sock.close()

        except AssertionError: #thrown by FileRequest.validate()

            print("[{}]  {}:{} RECIEVED MALFORMED PACKET".format(datetime.now().strftime(
                "%H:%M:%S"), client_address[0], client_address[1]))
            send_response(client_sock, None)
            client_sock.close()

        except socket.timeout:

            print("[{}]  {}:{} CONNECTION TIMED OUT".format(datetime.now().strftime(
                "%H:%M:%S"), client_address[0], client_address[1]))
            client_sock.close()


def main():
    run_server()


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

```
"""
Class to handle file request packets for socket based file downloader

3/8/21
Alex Burling
88866582
"""


import sys

MAGIC_NO = 0x497E


"""Abstracts request packet operations"""
class FileRequest:

    def __init__(self, magic_no, type, filename_len, filename):
        self.magic_no = magic_no
        self.type = type
        self.filename = filename
        self.filename_len = filename_len


    """Initialises a FileRequest object from a filename,
       used on the client side to generate the request"""
    @classmethod
    def from_filename(cls, filename):
        magic_no = MAGIC_NO
        type = 1
        filename = filename.encode('utf-8')
        filename_len = len(filename)

        if filename_len < 1 or filename_len > 1024:
            sys.exit("SPECIFIED FILENAME IS INVALID")

        return cls(magic_no, type, filename_len, filename)


    """Initialises a FileRequest object from a bytearray,
       used on the server side to reconstruct the request.
       Filename remains as a bytearray in case a complete
       request wasn't recieved in the first chunk"""
    @classmethod
    def from_bytearray(cls, array):
        magic_no = hex(int.from_bytes([array[0], array[1]], 'big'))
        type = int.from_bytes([array[2]], 'big')
        filename_len = int.from_bytes([array[3], array[4]], 'big')
        filename = bytearray(array[5:])

        return cls(magic_no, type, filename_len, filename)


    """Converts FileRequest data into a bytearray for
       transmission over socket"""
    def generate_packet(self):
        packet = bytearray()
        packet.extend(self.magic_no.to_bytes(2, 'big'))
        packet.extend(self.type.to_bytes(1, 'big'))
        packet.extend(self.filename_len.to_bytes(2, 'big'))
        packet.extend(self.filename)
        return packet
```

```
"""Validates packet structure on server side"""
def validate(self):
    if (self.magic_no != hex(MAGIC_NO)):
        raise AssertionError()
    if (self.type != 1):
        raise AssertionError()
    if (self.filename_len < 1 or self.filename_len > 1024):
        raise AssertionError()


"""Used to append data to FileRequest in case a complete request
wasn't recieved in the first chunk """
def append_data(self, data):
    self.filename.extend(data)


def check_len(self):
    return self.filename_len == len(self.filename)


def get_filename(self):
    return self.filename.decode("utf-8")
```

```
"""
Class to handle file response packets for socket based file downloader

3/8/21
Alex Burling
88866582
"""


import sys
import os

MAGIC_NO = 0x497E


"""Abstracts response packet operations"""
class FileResponse:

    def __init__(self, magic_no, type, status_code, data_len, data):
        self.magic_no = magic_no
        self.type = type
        self.status_code = status_code
        self.data_len = data_len
        self.data = data


    """Initialises a FileResponse object from a filename,
       used on the server side to generate the response"""
    @classmethod
    def from_filename(cls, filename):
        magic_no = MAGIC_NO
        type = 2
        if (filename == None): #If req validation throws an error
            status_code = 2
            data = ""
            data_len = 0
        elif (os.path.exists(filename)):
            status_code = 1
            file = open(filename, "rb")
            data = file.read()
            file.close()
            data_len = len(data)
        else: #Server can't find file
            status_code = 0
            data = ""
            data_len = 0
        return cls(magic_no, type, status_code, data_len, data)


    """Initialises a FileResponse object from a bytearray,
       used on the client side to reconstruct the response.
       Data remains as a bytearray in case a complete
       request wasn't recieved in the first chunk"""
    @classmethod
    def from_bytearray(cls, array):
        magic_no = hex(int.from_bytes([array[0], array[1]], 'big'))
        type = int.from_bytes([array[2]], 'big')
        status_code = int.from_bytes([array[3]], 'big')
        data_len = int.from_bytes([array[4], array[5], array[6], array[7]], 'big')
        data = bytearray(array[8:])
        return cls(magic_no, type, status_code, data_len, data)
```

```
"""Converts FileResponse data into a bytearray for
  transmission over socket and returns length for logging"""
def generate_packet(self):
    packet = bytearray()
    packet.extend(self.magic_no.to_bytes(2, 'big'))
    packet.extend(self.type.to_bytes(1, 'big'))
    packet.extend(self.status_code.to_bytes(1, 'big'))
    packet.extend(self.data_len.to_bytes(4, 'big'))
    packet.extend(self.data)
    packet_len = len(packet)
    return packet, packet_len


"""Validates packet structure on client side"""
def validate(self):
    if (self.magic_no != hex(MAGIC_NO)):
        sys.exit("RESPONSE PACKET MALFORMED")
    if (self.type != 2):
        sys.exit("RESPONSE PACKET MALFORMED")
    if (self.status_code == 0):
        sys.exit("SERVER COULDN'T FIND REQUESTED FILE")
    if (self.status_code == 2):
        sys.exit("SERVER RECIEVED MALFORMED PACKET")
    if (not self.check_len()):
        sys.exit("COULDN'T RECIEVE COMPLETE RESPONSE")


"""Used to append data to FileResponse in case a complete request
wasn't recieved in the first chunk """
def append_data(self, data):
    self.data.extend(data)


def check_len(self):
    return self.data_len == len(self.data)


def get_data(self):
    return self.data
```

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:	Alexander Robert Burling
Student ID:	88866582
Signature:	[illegible]
Date:	3/8/2021