COSC 264

Introduction to Computer Networks and the Internet

TCP Server/Client Assignment

Author: Shai Levin
Student ID: 59368709

```
        import socket
        import sys
        import datetime

        """
        A Simple TCP Server which accepts a port number as input and allows a client to download a file of
        its choosing from the server. The server communicates through stream
        / TCP sockets, exchanging both control and actual file data.

        Auth: Shai Levin
        Date: 15 August 2019
        """


        class Server:

            def __init__(self):
                """Initialises and execute the server."""
                self.host = socket.gethostbyname(socket.gethostname())  # Find the server's IP address
                self.setup()
                self.active()

            def setup(self):
                """Checks that the port number is valid and the only argument, and then attempts to open and ↵
        ↳       bind the socket"""
                if len(sys.argv) != 2:  # Checks that only 1 argument has been parsed into the command line
                    self.error("FATAL ERROR: More/less than one argument entered into command line")
                portnum = int(sys.argv[1])
                if portnum > 64000 or portnum < 1024:  # Checks port number falls in the allowed range
                    self.error("FATAL ERROR: Port number (%s) not within params" % portnum)
                try:
                    self.sock = socket.socket()
                    self.sock.bind((self.host, portnum))
                except:
                    self.sock.close()
                    self.error("FATAL ERROR: Creating and binding port socket")

            def active(self):
                """Active listening stage where the server enters an infinite loop and waits for file ↵
        ↳       requests. When it
                receives a connection it accepts it and performs file processing"""
                try:
                    self.sock.listen(100)
                except:
                    self.sock.close()
                    self.error("FATAL ERROR: Error establishing socket listen")

                while True:
                    current_socket, (host, port) = self.sock.accept()
                    print("%s : Connected to %s on port %s" % (datetime.datetime.now(), host, port))
                    current_socket.settimeout(1)
                    try:

                        header = self.header_process(current_socket.recv(5))
                        if header[0] != 0x497E or header[1] != 1 or header[2] > 1024 or header[2] < 1:
                            raise Exception
                        filename = current_socket.recv(header[2])
```

```
                    filetext = self.fileprocessing(filename)
                    fileresponse = self.packetcompose(filetext)

                    current_socket.send(fileresponse)
                except:
                    print("Error with FileRequest, closing current socket and resuming server operations")

                finally:
                    current_socket.close()

        def packetcompose(self, filetext):
            """Composes the File Response packet given the text of a file."""
            magicnumber = 0x497E
            type = 2
            statuscode = 0 if filetext is None else 1
            datalength = 0 if filetext is None else len(filetext)
            fileresponse = bytearray([magicnumber >> 8, magicnumber & 0xFF, type, statuscode,
                                      datalength >> 24, (datalength & 0xFF0000) >> 16,
                                      (datalength & 0xFF00) >> 8, datalength & 0xFF])
            if filetext is not None:
                fileresponse += filetext

            return fileresponse

        def fileprocessing(self, filename):
            """Reads the file 'filename' as a byte-string and returns it."""
            try:
                with open(filename, "rb") as file:  # Read file as a binary file.
                    # opens the file during processing, and is automatically closed after with block
                    text = file.read()
                    return text
            except:
                print("Error with opening and/or reading file, ensure file exists and has permission to ↵
    ↳           be opened")
                return None

        def header_process(self, byte_array):
            """Processes the file request header - a byte array containing 5 bytes
            - into a list of values for validation."""
            magicno = (byte_array[0] << 8) + byte_array[1]
            type = byte_array[2]
            filenamelen = (byte_array[3] << 8) + byte_array[4]
            return [magicno, type, filenamelen]

        def error(self, message):
            """Error function which prints the error and exits."""
            print(message)
            sys.exit()


    Server()
```

```
        import socket
        import sys
        import datetime
        import os

        """
        A Simple TCP Client which downloads a file of it's choosing from a server. The client accepts
        3 parameters: Destination Address, Port Number, and name of the file requested.

        Auth: Shai Levin
        Date: 15 August 2019
        """


        class Client:

            def __init__(self):
                """Initialises and executes the client program"""
                self.destination, self.port, self.filename = self.setup()
                self.sock = None  # Socket object defined in __init__ to obey style guides.
                self.socketprocessing()
                self.fileprocessing()

            def setup(self):
                """Validates arguments entered into the command line and returns
                the destination IP address, port and filename."""

                if len(sys.argv) != 4:  # Checks that exactly 3 arguments has been parsed into the command line
                    self.error("FATAL ERROR: More/less than one argument entered into command line")

                destination = sys.argv[1]
                try:
                    destination = socket.gethostbyname(destination)  # Ensures the destination address is in ↵
        ↳           IP format
                except:
                    self.error("FATAL ERROR: Unable to find IP address for \"%s\"" % sys.argv[1])

                portnum = int(sys.argv[2])
                if portnum > 64000 or portnum < 1024:  # Checks port number falls in the allowed range
                    self.error("FATAL ERROR: Port number (%s) not within params" % portnum)

                filename = sys.argv[3]
                if os.path.exists(filename):  # Checks file does not already exist to prevent over-writing.
                    self.error("FATAL ERROR: File \"%s\" already exists. Aborting to prevent over-writing" % ↵
        ↳           filename)

                return destination, portnum, filename

            def socketprocessing(self):
                """Opens and connects socket to server."""
                try:
                    self.sock = socket.socket()
                    self.sock.settimeout(1)  # Set timeout to prevent program from waiting too long in the ↵
        ↳           case of an error.
                    self.sock.connect((self.destination, self.port))
                except:
                    self.sock.close()
```

```
                self.error("FATAL ERROR: Error establishing socket connection")

        def fileprocessing(self):
            """Send the file request to the server and receives the file response.
            Then receives the file text and writes it to file."""
            try:
                filerequest = self.filerequest()
                self.sock.send(filerequest)

                fileresponse = self.fileresponse(self.sock.recv(8))  # Receive & process the 8 bytes of ↵
    ↳           the response header.
                if fileresponse is None:
                    raise Exception
                status, datalength = fileresponse

                # Catches the case where the server responds with a file request but the file was not ↵
    ↳           able to be read
                if status == 0:
                    self.error("FATAL ERROR: Server was not able to open requested file.")

                # Note this accounts for blank files, which are still allowed to be copied across from ↵
    ↳           the server.
                filedata = b""
                i = datalength
                while i > 4096:  # While loop to receive a maximum of 4096 bytes
                    filedata += self.sock.recv(4096)
                    i -= 4096
                filedata += self.sock.recv(i)

                if len(filedata) != datalength:  # Ensures the file data matches the length of the file ↵
    ↳           given by the header.
                    self.error("FATAL ERROR: File length (%s) does not match length given by header (%s)."
                               % (len(filedata), datalength))

                self.writefile(filedata)

                # Clean up and finish
                print("File \"%s\" of size %.2fkB received and saved" % (self.filename, (datalength/1000)))
                self.sock.close()

            except:
                self.sock.close()
                self.error("FATAL ERROR: Unable to communicate file response from server")

        def filerequest(self):
            """Generates the file request header to be transmitted to the server."""
            magicnumber = 0x497E
            type = 1
            filename = self.filename.encode('utf-8')
            filenamelen = len(filename)
            filerequest = bytearray([magicnumber >> 8, magicnumber & 0xFF, type, filenamelen >> 8,
                                     filenamelen & 0xFF]) + filename
            return filerequest

        def fileresponse(self, header):
            """Processes the file response header bytes and returns the status and length of the file ↵
    ↳       text."""
```

```
            magicno = (header[0] << 8) + header[1]
            type = header[2]
            status = header[3]
            datalength = (header[4] << 24) + (header[5] << 16) + (header[6] << 8) + header[7]
            if magicno != 0x497E or type != 2 or not (status == 1 or status == 0):
                return None
            return status, datalength

        def writefile(self, filedata):
            """Writes the file text to the file"""
            try:
                # 'with' block automatically closes the file after block.
                with open(self.filename, "wb+") as file: # Write file as a binary file.
                    file.write(filedata)
            except:
                os.remove(self.filename)  # Removes file if it fails to write.
                self.error("Error creating file \'%s\'" % self.filename)

        def error(self, message):
            """Error function which prints the error and exits."""
            print(message)
            sys.exit()


    Client()
```

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: Shai Levin

Student ID: 59368709

Signature: 

Date: 17/8/19