# Cover Sheet

Name: Jonathan Edwards (jme161)

Student-ID: 27033004

# 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: | Jonathan Edwards |
| Student ID: | 27033004 |
| Signature: | [illegible] |
| Date: | 16/08/2019 |

```python
import os
import time
import socket
from FileRequest import FileRequest, decodeFixedHeader
from FileResponse import FileResponse



# Fixed constants
BUFFER_SIZE = 10000


def currentTime():
    """
    Returns the current time
    """
    return time.strftime("%H:%M:%S", time.localtime())


def checkFile(soc, fd, fileName):
    """
    Attempting to read the file to see if it exists
    locally in the Server and is readable.
    - Returns Status Code of 1 when file exists and
      can be opened, otherwise 0.
    """
    errorMessage = "\n-ERROR: File doesn't exist locally in" \
        " the Server, closing and aborting...".encode('utf-8')
    successMessage =  "\n-SUCCESS: File does exist locally in" \
        " the Server, Transferring data now...".encode('utf-8')

    if not os.path.exists("Server/"+fileName):
        fd.send(errorMessage)
        fd.close()      # Closing th File Directory (fd) socket
        runServer(soc)  # Restating the loop process

    fd.send(successMessage)

    return True


def readFile(soc, fd, fileName, close):
    """
    Reading byte string data from the file.
    """
    try:
        fOpen = open("Server/"+fileName, 'rb')
        fRead = fOpen.readlines()
        if close:
            fOpen.close()
    except FileNotFoundError as err:
        print(str(err)+'\n')
        fd.close()      # Closing th File Directory (fd) socket
        runServer(soc)  # Restating the loop process

    return fRead


def sendResponse(soc, fd, fRead, fileName):
    """
    Sends byte data detailing the information the Client would like to
    retrieve from the Server.
    """
    record = bytearray(0)
    number = 0x497E
    dataLength = 0
    _type = 2

    # Gets the status code
    statusCode = checkFile(soc, fd, fileName)

    if statusCode == 0:
        dataLength = 0

    fr = FileResponse(number, statusCode, dataLength, _type)
    fr.encodeFixedHeader(record)

    # Concatenating string byte data with the 8 byte fixed header
    for line in fRead:
        record += line
        dataLength += len(line)

    fd.send(record)  # Sending file to Client

    # Clean up with message
    readFile(soc, fd, fileName, True)
    fd.close()      # Closing th File Directory (fd) socket

    if dataLength == 0:
        print("- Nothing to send from file.")
    else:
        print("A total of {0} bytes transferred succesfully " \
        "from the Server to the Client.\n".format(dataLength))
    runServer(soc)  # Restating the loop process


def fileRequest(soc, fd, data, startTime):
    """
    Checking to see if the server can can send the
    client a certain file if it exists.
    """
    # Checks the 5 byte fixed header
    (magicNum, _type, fileNameLen) = decodeFixedHeader(data)

    # If the time gap is greater then 1, restart process
    if (time.clock()-startTime) >= 1.0:
        print("\nERROR: File Request is erroneous, aborting...")
        print("Please try again.\n")
        fd.close()      # Closing th File Directory (fd) socket
        runServer(soc)  # Restating the loop process

    # Checking the validity of the File Request
    fr = FileRequest(magicNum, fileNameLen, _type)
    if fr.requestChecker():
        print("\nERROR: Couldn't read the record from the socket...")
        print("Please try again.\n")
        fd.close()      # Closing th File Directory (fd) socket
        runServer(soc)  # Restating the loop process

    fileName = data[5:].decode('utf-8')  # decoding the file name string byte data

    if len(fileName) != fileNameLen:
        print("\nERROR: The file could not be read properly...")
        print("Please try again.\n")
        fd.close()      # Closing th File Directory (fd) socket
        runServer(soc)  # Restating the loop process

    return fileName


def acceptSocket(soc):
    """
    Printing server acceptance message.
    """
    port = soc.getsockname()[1]
    fd, addr = soc.accept()
    print("-- {0}  IP = {1}  Port = {2}".format(currentTime(), addr[0], port))

    return fd


def setUpServer():
    """
    Checking for errors and setting up the server.
    """
    # Analysing the entered port number
    port = int(input("Please enter in a Port Number:\n>> "))
    if port < 1024 or 64000 < port:
        print("\nERROR: Port number '{0}' is not within values 1,024 and 64,000...".format(port))
        print("Terminating Program")
        exit()

    # Attempting to create a socket
    try:
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error as e:
        print('\n',str(e))
        exit()

    # Attempting to bind to the port number
    try:
        soc.bind(('', port))
    except socket.error as e:
        print('\n',str(e))
        exit()

    # Attempting to listen for the socket
    try:
        soc.listen(1)
    except socket.error as e:
        print('\n',str(e))
        soc.close()
        exit()

    return soc


def runServer(soc):
    """
    Runs the server until closed/exited.
    """
    while True:
        fd = acceptSocket(soc)
        startTime = time.clock()  # Start timer
        data = fd.recv(BUFFER_SIZE)  # Data sent from Client through a socket
        fileName = fileRequest(soc, fd, data, startTime)
        fRead = readFile(soc, fd, fileName, False)
        sendResponse(soc, fd, fRead, fileName)


def main():
    """
    Runs and Controls the program flow of the server.
    """
    soc = setUpServer()
    print("Waiting for Client to connect...\n")
    runServer(soc)


main()
```

```python
import os
import time
import socket
from FileRequest import FileRequest
from FileResponse import FileResponse, decodeFixedHeader


# Fixed constants
BUFFER_SIZE = 10000


def currentTime():
    """
    Returns the current time
    """
    return time.strftime("%H:%M:%S", time.localtime())


def writeFile(fileName, fileData):
    """
    Writing byte string data into the file.
    """
    fOpen = open("Client/"+fileName, 'w')
    fOpen.write(fileData)


def readResponse(soc, data, startTime, msgServer, fileName):
    """
    Read a Response from the Client.
    """
    fixedHeader = data[:8]  # Fixed Header byte array
    acturalData = data[8:]  # Actual Data byte array

    if len(data) <= 8:
        print("File doesnt exist")
        soc.close()  # Closing the socket
        exit()

    # Checks the first 8 bytes
    (magicNum, _type, statusCode, dataLength) = decodeFixedHeader(fixedHeader)

    # Determining the amount of recieved bytes
    if len(acturalData) == 0:
        print("\n- Noting was in the file.")
        soc.close()  # Closing the socket
        exit()
    else:
        print(msgServer)

    # If the time gap is greater then 1, terminate the program
    if (time.clock()-startTime) >= 1.0:
        print("\nERROR: File Response is erroneous...\nTerminating Program")
        soc.close()  # Closing the socket
        exit()

    # Checking the validity of the File Response
    fr = FileResponse(magicNum, statusCode, dataLength, _type)
    if fr.responseChecker():
        print("\nERROR: File Response is erroneous...\nTerminating Program")
        soc.close()  # Closing the socket
        exit()

    # Reading the actual data sent from Server
    if statusCode:
        writeFile(fileName, acturalData.decode('utf-8'))
        soc.close()  # Closing the socket
        exit()
    else:
        print("\nERROR: Unable to write the file you requested..." +
              "\nTerminating Program")
        soc.close()  # Closing the socket
        exit()
```

```python
def sendRequest(soc, fileName):
    """
    Sends byte data detailing the information the Client would like to
    retrieve from the Server.
    """
    record = bytearray(0)
    number = 0x497E

    fr = FileRequest(number, fileName)
    fr.encodeFixedHeader(record)

    # Sending Info to the Server
    soc.send(record)
    soc.send(fileName.encode('utf-8'))


def setUpClient():
    """
    Checking for errors and setting up the server.
    """
    # Analysing the entered host name, port number and file name
    try:
        (host, port, fileName) = input("Please enter in a Hosts Name, " +
        "Port Number and a File Name:\n>> ").split()
    except ValueError as e:
        print('\n',str(e))
        exit()

    try:
        addrInfo = socket.getaddrinfo(host, port)
    except socket.error as e:
        print('\n',str(e))
        exit()

    if int(port) < 1024 or 64000 < int(port):
        print("\nERROR: Port number '{0}' is not within values 1,024 and 64,000...".format(port))
        print("Terminating Program")
        exit()

    if os.path.exists("Client/"+fileName):
        print("\nERROR: File '{0}' already exists locally...\nTerminating Program.".format(fileName))
        exit()

    # Attempting to create a socket
    try:
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error as e:
        print('\n',str(e))
        exit()
    startTime = time.clock()  # Start timer

    # Attempting to connect with the server
    try:
        soc.connect(addrInfo[0][-1])
    except socket.error as e:
        print('\n',str(e))
        exit()

    return (soc, fileName, startTime)


def runClient():
    """
    Runs and Controls the program flow of the Client.
    """
    (soc, fileName, startTime) = setUpClient()
    sendRequest(soc, fileName)  # Sending a request
    msgServer = soc.recv(BUFFER_SIZE).decode('utf-8')  # Print server status on file

    # Attempting to read file
    if msgServer[2:7] != "ERROR":
        data = soc.recv(BUFFER_SIZE)  # Data sent from Client through a socket
        readResponse(soc, data, startTime, msgServer, fileName)  # Read response from server
    else:
        print(msgServer)
        soc.close()  # Closing the socket
        exit()


runClient()
```

```python
# Fixed constants
EXIT_SUCCESS = 0
EXIT_FAILURE = 1

BYTE_MASK = 0xFF
MAGIC_NO  = 0x497E  # required safeguard
TYPE      = 0x1     # required Type


class FileRequest():
    """
    Creating a file request.
    """

    def __init__(self, magicNum, fileName, _type=TYPE):
        """
        Init
        """
        self.magicNum = magicNum
        self._type = _type

        # Set the length of a file name
        try:
            self.fileNameLen = len(fileName)  # got a type string
        except TypeError:
            self.fileNameLen = fileName       # got a type int


    def encodeFixedHeader(self, record):
        """
        The Fixed Header is made up of 5 bytes. The Client
        sends these bytes over to the Server through the
        socket.
        - Stores byte informtion in a byte array.
        """
        # Encoding Fixed Header
        byte1 = self.magicNum >> 8
        byte2 = self.magicNum & BYTE_MASK
        byte3 = self._type
        byte4 = self.fileNameLen >> 8
        byte5 = self.fileNameLen & BYTE_MASK

        record += bytes([byte1]) + bytes([byte2]) + bytes([byte3]) + bytes([byte4]) + bytes([byte5])


    def requestChecker(self):
        """
        Checks the validity of the File Request record and returns the
        status of the Fixed Header.
        - Returns 0 if record is correct.
        """
        checkFileLen = self.fileNameLen < 1 or self.fileNameLen > 2**10
        if (self.magicNum != MAGIC_NO) or checkFileLen or (self._type != TYPE):
            return EXIT_FAILURE
        return EXIT_SUCCESS


def decodeFixedHeader(data):
    """
    Decodes the 5 byte Fixed Header and returns the three wanted
    values, (magicNum, _type and fileNameLen).
    """
    # Decoding Fixed Header
    magicNum = (data[0] << 8) | (data[1] & BYTE_MASK)
    _type = data[2]
    fileNameLen = (data[3] << 8) | (data[4] & BYTE_MASK)

    return (magicNum, _type, fileNameLen)
```

```python
# Fixed constants
EXIT_SUCCESS = 0
EXIT_FAILURE = 1

BYTE_MASK = 0xFF
MAGIC_NO  = 0x497E  # required safeguard
TYPE      = 0x2     # required Type


class FileResponse():
    """
    Creating a file request.
    """

    def __init__(self, magicNum, statusCode, dataLength, _type=TYPE):
        """
        Init
        """
        self.magicNum = magicNum
        self.statusCode = statusCode
        self.dataLength = dataLength
        self._type = _type


    def encodeFixedHeader(self, record):
        """
        The Fixed Header is made up of 8 bytes. The Client
        sends these bytes over to the Server through the
        socket.
        - Stores byte informtion in a byte array.
        """
        # Encoding Fixed Header
        byte1 = self.magicNum >> 8
        byte2 = self.magicNum & BYTE_MASK
        byte3 = self._type
        byte4 = self.statusCode
        byte5 = self.dataLength >> 24
        byte6 = (self.dataLength >> 16) & BYTE_MASK
        byte7 = (self.dataLength >> 8) & BYTE_MASK
        byte8 = self.dataLength & BYTE_MASK

        record += (bytes([byte1]) + bytes([byte2]) + bytes([byte3]) + bytes([byte4]) +
                   bytes([byte5]) + bytes([byte6]) + bytes([byte7]) + bytes([byte8]))


    def responseChecker(self):
        """

        """
        if ((self.magicNum != MAGIC_NO) or (self._type != TYPE) or
            (self.statusCode != 1 and self.statusCode != 0)):
            return EXIT_FAILURE
        return EXIT_SUCCESS


def decodeFixedHeader(data):
    """
    Decodes the 8 byte Fixed Header and returns the three wanted
    values, (magicNum, _type and fileNameLen).
    """
    # Decoding Fixed Header
    magicNum = (data[0] << 8) | (data[1] & BYTE_MASK)
    _type = data[2]
    statusCode = data[3]
    dataLength = ((data[4] << 24) | ((data[5] << 16) & BYTE_MASK) |
                 ((data[6] << 8) & BYTE_MASK) | (data[7] & BYTE_MASK))

    return (magicNum, _type, statusCode, dataLength)
```