**Network Programming for Engineers (ECE 5650)**

**Project 1**

**Team Members Names: Anika Tasnim & Li Lin**

**Source Code(s):**

**Source code for server.py:**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# File Name：Server.py

# Created: 9/16/2020

# Author: Li Lin & Anika Tasnim

import os

import sys

import socket

import threading

import tkinter as tk

import tkinter.scrolledtext as ScrolledText

from datetime import datetime

import time

import struct

import re

serverPort = 12000

#Create main window

window = tk.Tk()

#Set widnow's title

window.title('Server')

#Set window width and height

window.geometry('1040x650')

LoggingText = ScrolledText.ScrolledText(window, height=10, width = 68)

LoggingText.place(x=200, y=30, anchor='nw')

ReceivedText = ScrolledText.ScrolledText(window, height=33, width = 68)

ReceivedText.place(x=10, y=200, anchor='nw')

ProcessedText = ScrolledText.ScrolledText(window, height=33, width = 68)

ProcessedText.place(x=520, y=200, anchor='nw')

#Labels

LogLabel = tk.Label(window, text='Log')

LogLabel.place(x=200, y=5, anchor='nw')

ReceivedLabel = tk.Label(window, text='Received')

ReceivedLabel.place(x=20, y=175, anchor='nw')

ProcessedLabel = tk.Label(window, text='Processed')

ProcessedLabel.place(x=530, y=175, anchor='nw')

isConnected = False

def StartServerThread():

    global isConnected

    try:

        while True:

            if isConnected ==False:

                #Create socket with IPv4/TCP type

                with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as serversocket:

                    #bind with server address and port

                    serversocket.bind(('',serverPort))

                    #Start to monitor

                    serversocket.listen(1)

                    LoggingText.insert('insert', 'Waiting for connection\n')

                    #wait for client's connection

                    connection,addr = serversocket.accept()

                    #print(connection, addr)

                    LoggingText.insert('insert', 'connected with {0}:{1}\n'.format(addr[0],addr[1]))

                    LoggingText.insert('insert', 'Waiting for request\n')

                    isConnected = True

                    rcv\_thread = threading.Thread(target=ReceiveDataThread, name = 'ReceiveDataThread', args=(connection,addr), daemon=True)

                    if not rcv\_thread.is\_alive():

                        rcv\_thread.start()

    except socket.error as msg:

        print(msg)

    else:

        print("Start thread finished")

def UpdateLoggingToEnd():

    while True:

        LoggingText.see('end')

        time.sleep(0.5)

def StartServer():

    start\_thread = threading.Thread(target=StartServerThread, name = 'StartServerThread', daemon=True)

    if not start\_thread.is\_alive():

        start\_thread.start()

        print('Start threading started')

    else:

        LoggingText.insert('insert', 'server is already started\n')

    UpdateLogging\_thread = threading.Thread(target=UpdateLoggingToEnd, name = 'UpdateLoggingthread', daemon=True)

    if not UpdateLogging\_thread.is\_alive():

        UpdateLogging\_thread.start()

StartButton = tk.Button(window, text='Start', font=('Arial',12), width=14, height=4, command = StartServer)

StartButton.place(x=30, y=30, anchor='nw')

#reqtype = ('SEARCH','REPLACE','REVERSE')

def ReceiveDataThread(connection,address):

    global isConnected

    request = ''

    search\_word = ''

    replace\_word = ''

    with connection:

        status = 'WAIT\_FOR\_REQUEST'

        while isConnected:

            print('waiting receive data')

            if(status == 'WAIT\_FOR\_REQUEST'):

                #Receive message from client

                message = connection.recv(1024)

                if not message:

                    isConnected = False

                    break

                message = message.decode()

                print(message)

                if message.find('SEARCH+') != -1:

                    LoggingText.insert('insert', 'Search request received and accepted\n')

                    request = 'SEARCH'

                    search\_word = message.split('+',1)[1]

                    if search\_word == '':

                        message = 'No search word defined!'

                        #keep current status

                    else:

                        message = 'Search request accepted'

                        status = 'WAIT\_FOR\_FILE\_INFO'

                    connection.send(message.encode())

                elif message.find('REPLACE+') != -1:

                    LoggingText.insert('insert', 'Replace request received and accepted\n')

                    request = 'REPLACE'

                    msg\_list = message.split('+',2)

                    search\_word = msg\_list[1]

                    replace\_word = msg\_list[2]

                    if search\_word == '':

                        message = 'No search word defined!'

                        #keep current status

                    else:

                        message = 'Replace request accepted'

                        status = 'WAIT\_FOR\_FILE\_INFO'

                    connection.send(message.encode())

                elif message =='REVERSE':

                    LoggingText.insert('insert', 'reverse request received and accepted\n')

                    request = 'REVERSE'

                    message = 'Reverse request accepted'

                    connection.send(message.encode())

                    status = 'WAIT\_FOR\_FILE\_INFO'

                elif message == 'EXIT':

                    LoggingText.insert('insert', 'Exit request received\n')

                    connection.close()

                    isConnected = False

                else:

                    message ='unrecognized request!'

            elif (status == 'WAIT\_FOR\_FILE\_INFO'):

                fileinfo\_size = struct.calcsize('128sQ')

                fileinfo\_data = connection.recv(fileinfo\_size)

                if not fileinfo\_data:

                    isConnected = False

                    break

                #Receive file name and size info

                filename,filesize = struct.unpack('128sQ',fileinfo\_data)

                rcv\_file\_name = filename.decode('utf-8').strip('\x00')

                LoggingText.insert('insert', 'Head info received\n')

                #Receive the data of file

                received\_size = 0

                all\_data\_str = ''

                with open(rcv\_file\_name, 'wb') as rcv\_file\_handle:

                    #Clear the content firstly

                    ReceivedText.delete(1.0,'end')

                    while not (received\_size == filesize):

                        if(filesize - received\_size > 1024):

                            data = connection.recv(1024)

                            if not data:

                                isConnected = False

                                break

                            received\_size += len(data)

                        else:

                            data = connection.recv(filesize - received\_size)

                            received\_size = filesize

                        rcv\_file\_handle.write(data)

                        ReceivedText.insert('insert',data.decode())

                        all\_data\_str = all\_data\_str + data.decode()

                    if isConnected == False:

                        LoggingText.insert('insert', '{0} file transfer failed\n'.format(rcv\_file\_name))

                    else:

                        LoggingText.insert('insert', 'Received all data of {0}\n'.format(rcv\_file\_name))

                        ProcessedText.delete(1.0,'end')

                        #Process the file according to request

                        if request == 'SEARCH':

                            #Search

                            count = all\_data\_str.count(search\_word)

                            message = 'There are {0} words "{1}" found in {2}.'.format(count,search\_word,rcv\_file\_name)

                            ProcessedText.insert('insert','There are {0} words "{1}" found in {2}'.format(count,search\_word,rcv\_file\_name))

                            connection.send(message.encode())

                            status = 'WAIT\_FOR\_REQUEST'

                            LoggingText.insert('insert', 'Search result sent\n')

                        elif request == 'REPLACE':

                            #Replace

                            replaced\_data = all\_data\_str.replace(search\_word,replace\_word)

                            replaced\_file\_name = os.path.join('./', 'Replaced\_' + rcv\_file\_name)

                            ProcessedText.insert('insert',replaced\_data)

                            #Store local file

                            with open(replaced\_file\_name, 'wb') as new\_file\_handle:

                                new\_file\_handle.write(replaced\_data.encode())

                            with open(replaced\_file\_name, 'rb') as new\_file\_handle:

                                #Send file info to client

                                fileinfo\_size = struct.calcsize('128sQ')    #file name lentgh = 128 bytes; filesize = 8bytes

                                #define file head info, including name and size

                                fhead = struct.pack('128sQ', bytes(replaced\_file\_name.encode('utf-8')), len(replaced\_data.encode('utf-8')))

                                connection.send(fhead)

                                LoggingText.insert('insert', 'Replaced file header info sent\n')

                                #send file data to client

                                while True:

                                    send\_data = new\_file\_handle.read(1024)

                                    print(send\_data.decode())

                                    if not send\_data:

                                        LoggingText.insert('insert', 'Replaced file send over...\n')

                                        break

                                    connection.send(send\_data)

                                    print(data.decode())

                            status = 'WAIT\_FOR\_REQUEST'

                        elif request == 'REVERSE':

                            #Reverse

                            data\_str\_list = all\_data\_str.split()

                            reversed\_data = ' '.join(reversed(data\_str\_list))

                            ProcessedText.insert('insert',reversed\_data)

                            reversed\_file\_name = os.path.join('./', 'Replaced\_' + rcv\_file\_name)

                            #Store local file

                            with open(reversed\_file\_name, 'wb') as new\_file\_handle:

                                new\_file\_handle.write(reversed\_data.encode())

                            with open(reversed\_file\_name, 'rb') as new\_file\_handle:

                                #Send file info to client

                                fileinfo\_size = struct.calcsize('128sQ')    #file name lentgh = 128 bytes; filesize = 8bytes

                                #define file head info, including name and size

                                fhead = struct.pack('128sQ', bytes(reversed\_file\_name.encode('utf-8')), len(reversed\_data.encode('utf-8')))

                                connection.send(fhead)

                                LoggingText.insert('insert', 'Reversed file header info sent\n')

                                #send file data to client

                                while True:

                                    send\_data = new\_file\_handle.read(1024)

                                    if not send\_data:

                                        LoggingText.insert('insert', 'Reversed file send over...\n')

                                        break

                                    connection.send(send\_data)

                            status = 'WAIT\_FOR\_REQUEST'

                        else:

                            message = 'unrecognized request!'

                            status = 'WAIT\_FOR\_REQUEST'

            else:

                print('Server is in unknown status')

                status = 'WAIT\_FOR\_REQUEST'

    LoggingText.insert('insert', 'Connection closed\n')

window.mainloop()

**Source code for client.py:**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# File Name：Client.py

# Created: 9/16/2020

# Author: Li Lin & Anika Tasnim

import socket

import tkinter as tk

from tkinter import filedialog

from  tkinter  import ttk

import tkinter.scrolledtext as ScrolledText

import os

import sys

import struct

import threading

import win32api

from datetime import datetime

import time

serverName = 'localhost'    #'192.168.0.15'

serverPort = 12000

CurrentDirectory = os.getcwd()

#Create main GUI window

window = tk.Tk()

#Set widnow's title

window.title('Client')

#Set window width and height

window.geometry('1305x650')

isConnected = False

#Create a Entry

SearchWordVar = tk.StringVar()

SearchWordEntry = tk.Entry(window, show=None, font=('Arial',14), width = 12, textvariable=SearchWordVar)

SearchWordEntry.place(x=150, y=80, anchor='nw')

SearchWordVar.set('Mobile')

ReplaceWordVar = tk.StringVar()

ReplaceWordEntry = tk.Entry(window, show=None, font=('Arial',14), width = 12, textvariable=ReplaceWordVar)

ReplaceWordEntry.place(x=300, y=80, anchor='nw')

ReplaceWordVar.set('iPhone')

SourceFilePathVar = tk.StringVar()

SourceFilePathEntry = tk.Entry(window, show=None, font=('Arial',14), width = 44, textvariable=SourceFilePathVar)

SourceFilePathEntry.place(x=10, y=160, anchor='nw')

SourceFilePathVar.set('C:/GitHubProject/NetworkProgramming/Project1/Mobile IP wiki.txt')

SaveFileNameVar = tk.StringVar()

SaveFileNameEntry = tk.Entry(window, show=None, font=('Arial',14), width = 15, textvariable=SaveFileNameVar)

SaveFileNameEntry.place(x=1000, y=194, anchor='nw')

SaveFileNameVar.set('output.txt')

#Label

LogLabel = tk.Label(window, text='Log')

LogLabel.place(x=650, y=10, anchor='nw')

SourceLabel = tk.Label(window, text='Source file')

SourceLabel.place(x=20, y=210, anchor='nw')

ReceivedLabel = tk.Label(window, text='Received from server')

ReceivedLabel.place(x=670, y=210, anchor='nw')

SearchLabel = tk.Label(window, text='Search word')

SearchLabel.place(x=150, y=59, anchor='nw')

ReplaceLabel = tk.Label(window, text='Replace word')

ReplaceLabel.place(x=300, y=59, anchor='nw')

SaveLabel = tk.Label(window, text='Save file name')

SaveLabel.place(x=1000, y=173, anchor='nw')

#Create Text

SourceFileText = ScrolledText.ScrolledText(window, height=30, width = 88)

SourceFileText.place(x=10, y=230, anchor='nw')

if os.path.isfile(SourceFilePathVar.get()):

    with open(SourceFilePathVar.get(),'rb') as reader:

        SourceFileText.delete(1.0,'end')

        text = reader.read()

        SourceFileText.insert('insert',text)

ProcessedFileText = ScrolledText.ScrolledText(window, height=30, width = 90)

ProcessedFileText.place(x=650, y=230, anchor='nw')

LoggingText = ScrolledText.ScrolledText(window, height=10, width = 90)

LoggingText.place(x=650, y=40, anchor='nw')

#update text to show end content

def UpdateLoggingToEnd():

    while True:

        LoggingText.see('end')

        time.sleep(0.5)

UpdateLogging\_thread = threading.Thread(target=UpdateLoggingToEnd, name = 'UpdateLoggingthread', daemon=True)

if not UpdateLogging\_thread.is\_alive():

    UpdateLogging\_thread.start()

#Socket created

clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#Create buttons

def ConnectServer():

    global isConnected

    if not isConnected:

        try:

            #Setup connection with server

            clientSocket.connect((serverName,serverPort))

        except socket.error as msg:

            now = str(datetime.now())[:-7]

            LoggingText.insert('insert','{0}: Server Connected failed({1})\n'.format(now,msg))

        else:

            isConnected = True

            now = str(datetime.now())[:-7]

            LoggingText.insert('insert','{0}: Server Connected\n'.format(now))

    else:

        LoggingText.insert('insert','Server already Connected )\n')

ConnectButton = tk.Button(window, text='Connect', font=('Arial',12), width=10, height=2, command = ConnectServer)

ConnectButton.place(x=10, y=10, anchor='nw')

def ExitThread():

    global isConnected

    if isConnected:

        request = 'EXIT'

        clientSocket.send(request.encode())

        LoggingText.insert('insert','Exit request sent to server\n')

        clientSocket.close()

    else:

        LoggingText.insert('insert','No connection\n')

    window.destroy()

def ExitProcess():

    exit\_thread = threading.Thread(target=ExitThread, name='ExitThread')

    exit\_thread.setDaemon(True)

    exit\_thread.start()

    print('Exit threading started')

ExitButton = tk.Button(window, text='Exit', font=('Arial',12), width=10, height=2, command = ExitProcess)

ExitButton.place(x=10, y=70, anchor='nw')

def SearchThread():

    global isConnected

    if isConnected:

        #1.Send request to server

        search\_word = SearchWordVar.get()

        request = 'SEARCH+' + search\_word

        clientSocket.send(request.encode())

        LoggingText.insert('insert','Search request sent with search word "{0}"\n'.format(search\_word))

        #Receive message from server

        response = clientSocket.recv(1024)

        if response:

            LoggingText.insert('insert', 'Response from server: {0} \n'.format(response.decode('utf-8')))

            if response.decode() == 'Search request accepted':

                filepath = SourceFilePathVar.get()

                if os.path.isfile(filepath):

                    #2. Send file info to server

                    fileinfo\_size = struct.calcsize('128sQ')    #file name lentgh = 128 bytes; filesize = 8bytes

                    #define file head info, including name and size

                    fhead = struct.pack('128sQ', bytes(os.path.basename(filepath).encode('utf-8')),

                                            os.stat(filepath).st\_size)

                    clientSocket.send(fhead)

                    LoggingText.insert('insert', 'File header sent\n')

                    #3. Send data to server

                    with open(filepath, 'rb') as fp:

                        while 1:

                            data = fp.read(1024)

                            if not data:

                                LoggingText.insert('insert', 'file send over...\n')

                                break

                            clientSocket.send(data)

                    #4. Receive the search result

                    response = clientSocket.recv(1024)

                    if response:

                        #5. Display the search result

                        LoggingText.insert('insert', 'Search result received\n')

                        ProcessedFileText.delete(1.0,'end')

                        ProcessedFileText.insert('insert',response.decode())

                    else:

                        isConnected = False

                        LoggingText.insert('insert', 'Server connection closed! Please check if server is still running\n')

                else:

                    LoggingText.insert('insert','The file path is not valid')

        else:

            isConnected = False

            LoggingText.insert('insert', 'Server connection closed! Please check if server is still running\n')

    else:

        LoggingText.insert('insert', 'No connection! Please connect firstly\n')

def ReplaceThread():

    global isConnected

    if isConnected:

        #1.Send request to server

        search\_word = SearchWordVar.get()

        replace\_word = ReplaceWordVar.get()

        request = 'REPLACE+' + search\_word +'+' + replace\_word

        clientSocket.send(request.encode())

        LoggingText.insert('insert','Replace request sent with  search word "{0}" and replace word "{1}"\n'.format(search\_word,replace\_word))

        #Receive message from server

        response = clientSocket.recv(1024)

        if response:

            LoggingText.insert('insert', 'Response from server: {0} \n'.format(response.decode('utf-8')))

            if response.decode() == 'Replace request accepted':

                filepath = SourceFilePathVar.get()

                if os.path.isfile(filepath):

                    #2. Send file info to server

                    fileinfo\_size = struct.calcsize('128sQ')    #file name lentgh = 128 bytes; filesize = 8bytes

                    #define file head info, including name and size

                    fhead = struct.pack('128sQ', bytes(os.path.basename(filepath).encode('utf-8')),

                                            os.stat(filepath).st\_size)

                    clientSocket.send(fhead)

                    LoggingText.insert('insert', 'File header sent\n')

                    #3. Send data to server

                    with open(filepath, 'rb') as fp:

                        while 1:

                            data = fp.read(1024)

                            if not data:

                                LoggingText.insert('insert', 'file send over...\n')

                                break

                            clientSocket.send(data)

                    #4. Receive the replace result

                    fileinfo\_size = struct.calcsize('128sQ')

                    fileinfo\_data = clientSocket.recv(fileinfo\_size)

                    if fileinfo\_data:

                        filename,filesize = struct.unpack('128sQ',fileinfo\_data)

                        rcv\_file\_name = filename.decode('utf-8').strip('\x00')

                        LoggingText.insert('insert', '{0} header info is received and size is {1} bytes\n'.format(rcv\_file\_name,filesize))

                        received\_size = 0

                        received\_data = ''

                        while not (received\_size == filesize):

                            if(filesize - received\_size > 1024):

                                data = clientSocket.recv(1024)

                                if data:

                                    received\_size += len(data)

                                else:

                                    isConnected = False

                                    break

                            else:

                                data = clientSocket.recv(filesize - received\_size)

                                if data:

                                    received\_size = filesize

                                else:

                                    isConnected = False

                                    break

                            received\_data = received\_data + data.decode()

                        if isConnected:

                            LoggingText.insert('insert', 'Replaced file {0} is received\n'.format(rcv\_file\_name))

                            #5. Display the replaced result

                            ProcessedFileText.delete(1.0,'end')

                            ProcessedFileText.insert('insert', received\_data)

                        else:

                            LoggingText.insert('insert', 'Server connection closed! Please check if server is still running\n')

                else:

                    LoggingText.insert('insert', 'The file path is not valid\n')

            else:

                isConnected = False

                LoggingText.insert('insert', 'Server connection closed! Please check if server is still running\n')

    else:

        LoggingText.insert('insert', 'No connection! Please connect firstly\n')

def ReverseThread():

    global isConnected

    if isConnected:

        #1.Send request to server

        request = 'REVERSE'

        clientSocket.send(request.encode())

        LoggingText.insert('insert','Reverse request sent\n')

        #Receive message from server

        response = clientSocket.recv(1024)

        if response:

            LoggingText.insert('insert', 'Response from server: {0} \n'.format(response.decode('utf-8')))

            if response.decode() == 'Reverse request accepted':

                filepath = SourceFilePathVar.get()

                if os.path.isfile(filepath):

                    #2. Send file info to server

                    fileinfo\_size = struct.calcsize('128sQ')    #file name lentgh = 128 bytes; filesize = 8bytes

                    #define file head info, including name and size

                    fhead = struct.pack('128sQ', bytes(os.path.basename(filepath).encode('utf-8')),

                                            os.stat(filepath).st\_size)

                    clientSocket.send(fhead)

                    LoggingText.insert('insert', 'File header sent\n')

                    #3. Send data to server

                    with open(filepath, 'rb') as fp:

                        while 1:

                            data = fp.read(1024)

                            if not data:

                                LoggingText.insert('insert', 'file send over...\n')

                                break

                            clientSocket.send(data)

                    #4. Receive the reversed result

                    fileinfo\_size = struct.calcsize('128sQ')

                    fileinfo\_data = clientSocket.recv(fileinfo\_size)

                    if fileinfo\_data:

                        filename,filesize = struct.unpack('128sQ',fileinfo\_data)

                        LoggingText.insert('insert', 'file header info is received\n')

                        received\_size = 0

                        received\_data = ''

                        while not (received\_size == filesize):

                            if(filesize - received\_size > 1024):

                                data = clientSocket.recv(1024)

                                if data:

                                    received\_size += len(data)

                                else:

                                    isConnected = False

                                    break

                            else:

                                data = clientSocket.recv(filesize - received\_size)

                                if data:

                                    received\_size = filesize

                                else:

                                    isConnected = False

                                    break

                            received\_data = received\_data + data.decode()

                        if isConnected:

                            LoggingText.insert('insert', 'Reversed file is received\n')

                            #5. Display the replaced result

                            ProcessedFileText.delete(1.0,'end')

                            ProcessedFileText.insert('insert', received\_data)

                        else:

                            LoggingText.insert('insert', 'No connection! Please connect firstly\n')

                else:

                    LoggingText.insert('insert','The file path is not valid')

        else:

            isConnected = False

            LoggingText.insert('insert', 'No connection! Please connect firstly\n')

    else:

        LoggingText.insert('insert', 'No connection! Please connect firstly\n')

def SearchWordFromServer():

    search\_thread = threading.Thread(target=SearchThread, name='SearchThread')

    search\_thread.setDaemon(True)

    search\_thread.start()

    print('Search threading started')

SearchButton = tk.Button(window, text='Search', font=('Arial',12), width=14, height=2, command = SearchWordFromServer)

SearchButton.place(x=150, y=5, anchor='nw')

def ReplaceWordByServer():

    replace\_thread = threading.Thread(target=ReplaceThread, name='replace\_thread')

    replace\_thread.setDaemon(True)

    replace\_thread.start()

    print('Replace threading started')

ReplaceButton = tk.Button(window, text='Replace', font=('Arial',12), width=14, height=2, command = ReplaceWordByServer)

ReplaceButton.place(x=300, y=5, anchor='nw')

def ReverseWordByServer():

    reverse\_thread = threading.Thread(target=ReverseThread, name='reverse\_thread')

    reverse\_thread.setDaemon(True)

    reverse\_thread.start()

    print('Reverse threading started')

ReverseButton = tk.Button(window, text='Reverse', font=('Arial',12), width=14, height=2, command = ReverseWordByServer)

ReverseButton.place(x=450, y=5, anchor='nw')

def SelectFile():

    SourceFilePath = filedialog.askopenfilename(title='Select source file', filetypes=[("Text files", "\*.txt"), ("all files", "\*.\*")])

    SourceFilePathEntry.delete(0,tk.END)

    SourceFilePathEntry.insert(0,SourceFilePath)

    with open(SourceFilePathVar.get(),'rb') as reader:

        SourceFileText.delete(1.0,'end')

        text = reader.read()

        SourceFileText.insert('insert',text)

SelectButton = tk.Button(window, text='Source path...', font=('Arial',12), width=12, height=1, command = SelectFile)

SelectButton.place(x=510, y=157, anchor='nw')

#callback function for Save button

def SaveFile():

    filename = SaveFileNameVar.get()

    filepath = os.path.join(CurrentDirectory, filename)

    print(filepath)

    with open(filepath, 'w') as fp:

        file\_data = ProcessedFileText.get(1.0,'end')

        fp.write(file\_data)

    fp.close()

    LoggingText.insert('end', 'File {0} saved under {1}\n'.format(filename,CurrentDirectory))

#Create Save button

SaveButton = tk.Button(window, text='Save', font=('Arial',12), width=10, height=1, command = SaveFile)

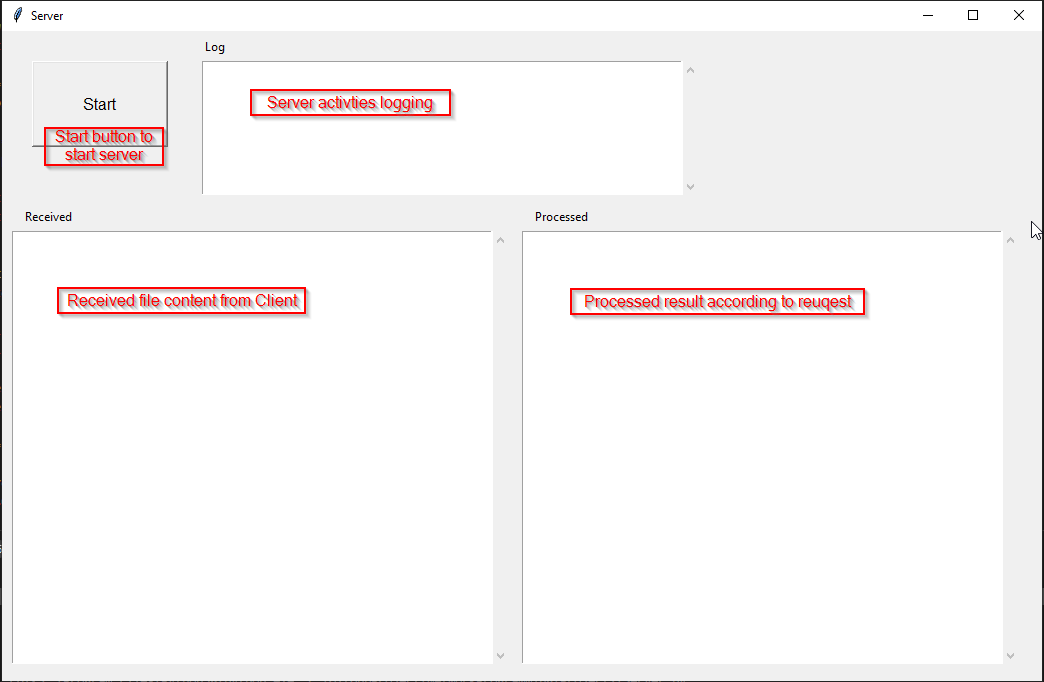
SaveButton.place(x=1175, y=190, anchor='nw')

window.mainloop()

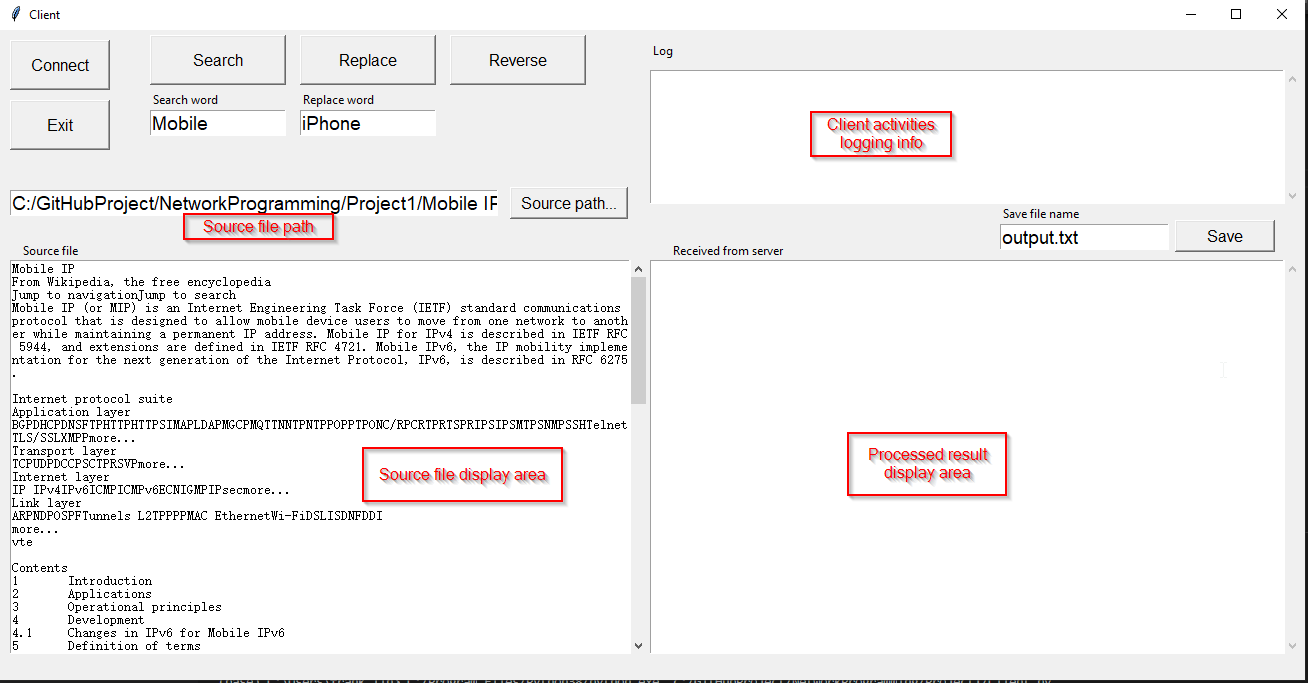
clientSocket.close()

**Overview of the tools:**

1. Server program GUI overview.

****

1. Client program GUI overview.

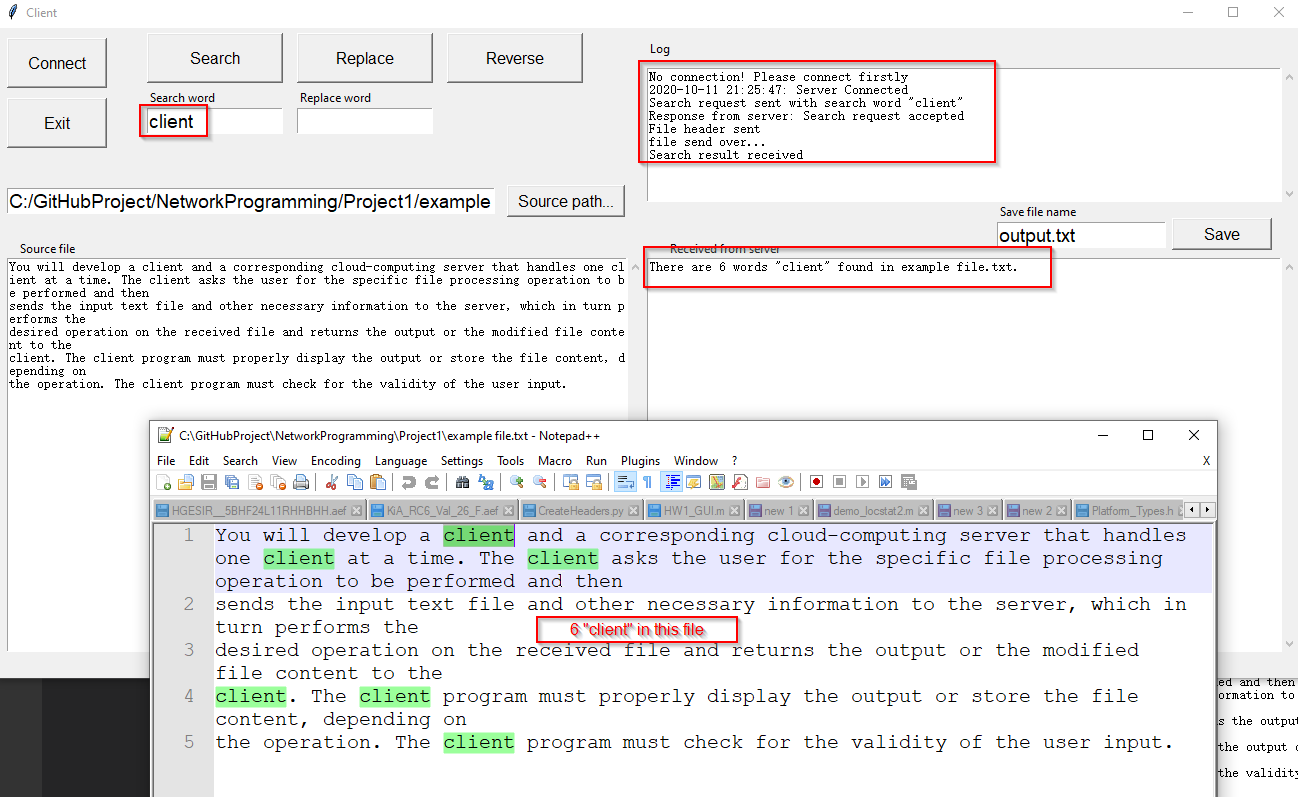
****

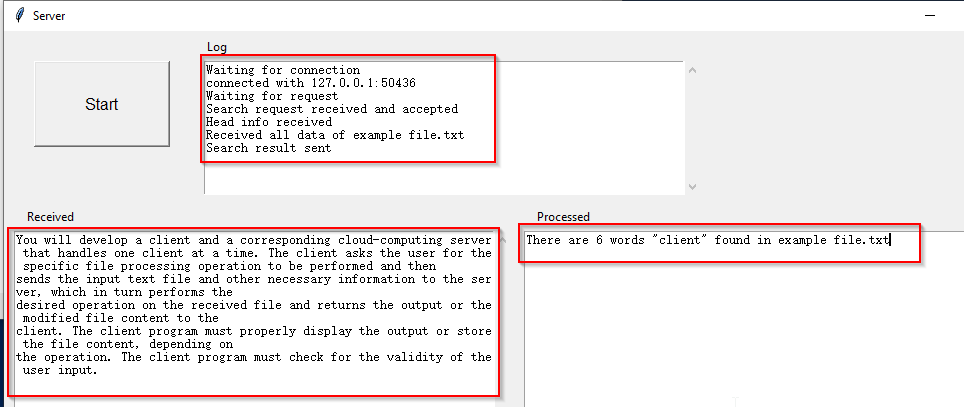
**Testing Procedure, including Description of Inputs**

1. Start Server program, and click on “Start” button.
   1. Server log info should show “Waiting for connection”
2. Start Client program, and click on “Connect” button.
   1. Client log info should show “Server Connected
   2. Server log info should show which client connected and from which port.
   3. Server should wait for a client’s request.
3. In the Client program, do below steps for each function:
   1. **Searching function**:
      1. Click on “Source path…” button to select the file you want to process, the file content shall be displayed in “Source file display area”
      2. Enter the searching word in “Searching word” entry area
      3. Click on button “Search”
      4. Check the result on “Processed result display area” of both Client and Server, they are should be same.
      5. Check on Server program window, the received file content should be same as the Client’s source file.
      6. Check log info of both Client and Server. They should show detail communication steps of search function.
   2. **Replacing function**:
      1. Click on “Source path…” button to select the file you want to process.(skip this step if you want to keep processing previous file loaded)
      2. Enter the searching word in “Searching word” entry area
      3. Enter the replacing word in “Replacing word” entry area
      4. Click on button “Replace”
      5. Check on Server program window, the Received file content should be same as the Client’s source file
      6. Check the Received from server area which should be same as Server Processed area.
      7. Enter the file name in “Save file name” area, Click on “Save” button to save the file.
      8. Check if the file is saved under the location same as Client program.
      9. Check log info of both Client and Server. They should show detail communication steps of Replace function.
   3. **Reverse function**:
      1. Click on “Source path…” button to select the file you want to process.(Skip this step if you want to keep processing previous file loaded)
      2. Click on button “Reverse”
      3. In Client window, Check the result on “Received from server” area which should be same as server’s Processed area content.
      4. Check on Server program window, the Received file content should be same as the Client’s source file.
      5. Enter the file name in “Save file name” area, Click on “Save” button to save the file.
      6. Check if the file is saved under the location same as Client program.
      7. Check log info of both Client and Server. They should show detail communication steps of Replace function.
   4. **Display function**:
      1. Display function is already shown in above steps
      2. After select the source file via “Source path…”, the source file content shall be displayed in below display area automatically.
      3. The received the result shall be displayed in “Processed result display area” automatically.
   5. **Exit function**:
      1. After any step above, click on “Exit” button
      2. Client should send “EXIT” request to Server if there is connection with server; Server should received this request and close current connection, then wait for another connection.
      3. Client program should be closed.

**Screenshots and Their Explanations:**

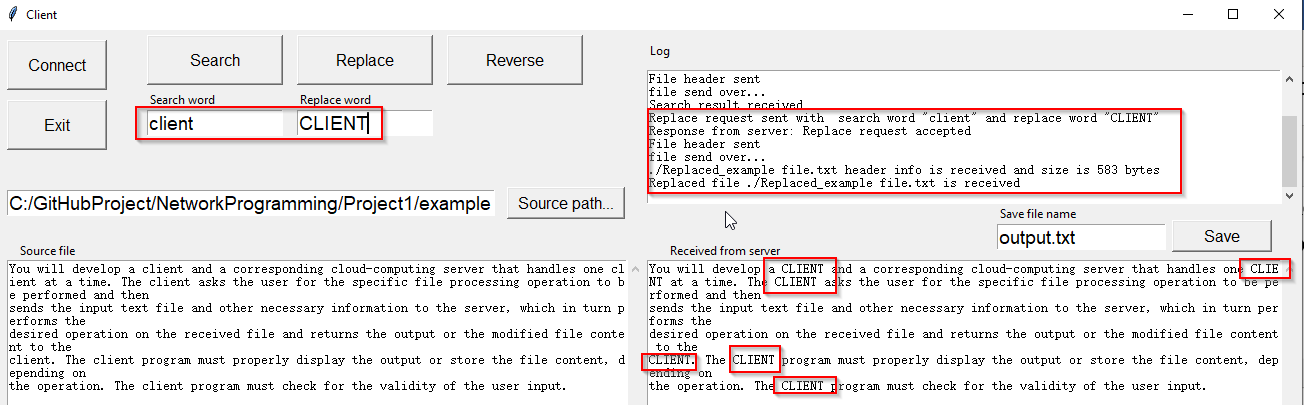
1. **Search function:**

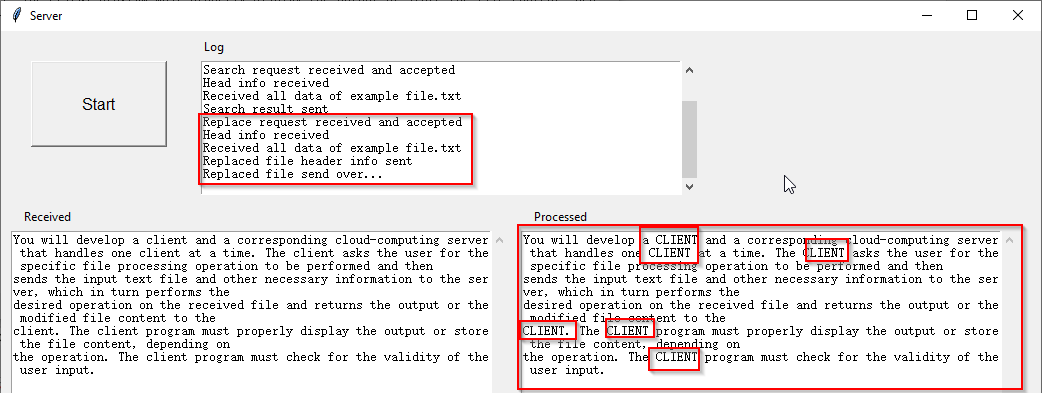
****

****

* + From above figure, the word “client” is sent to server and server accepted the request. And server sent “There are 6 words “client” found in example file.txt.
  + The info in Server program matches with Client’s
  + Log info is also showing detail steps of the communication.

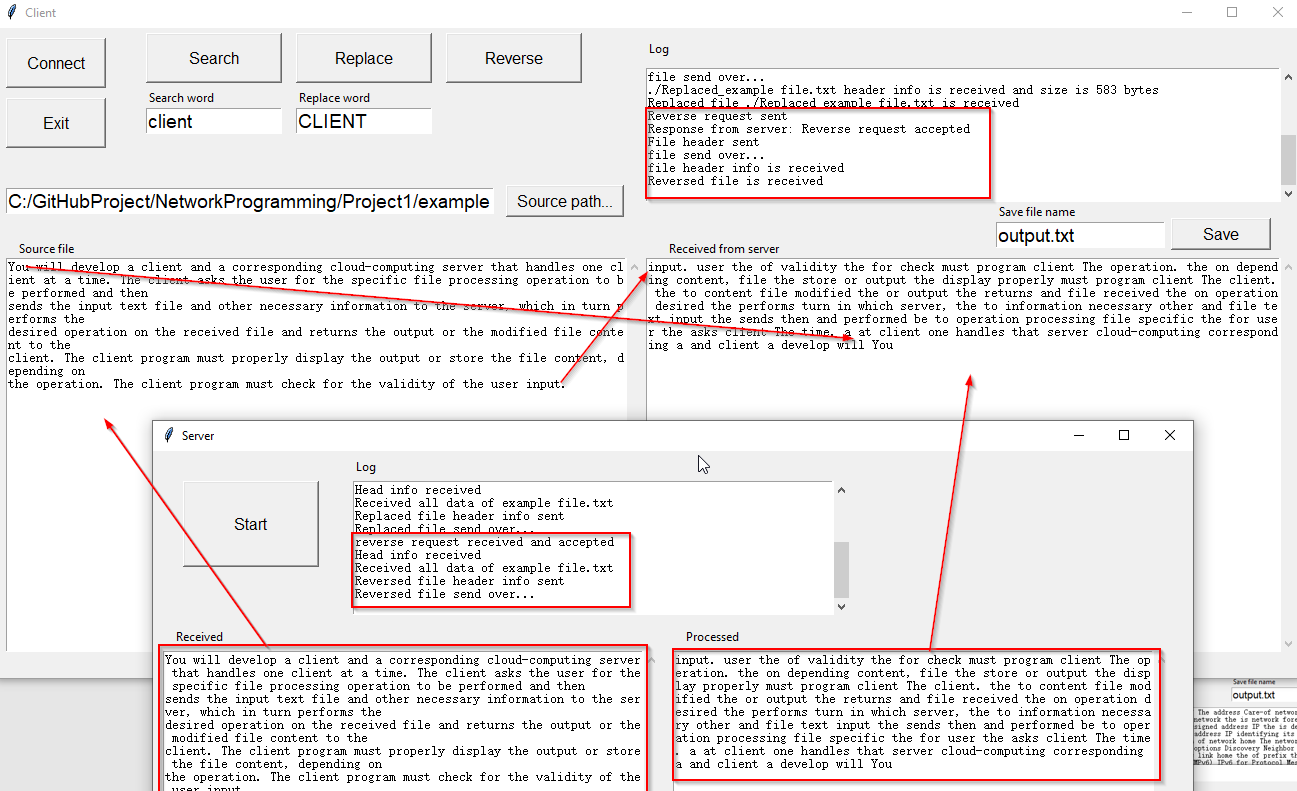
1. **Replace function**

****

****

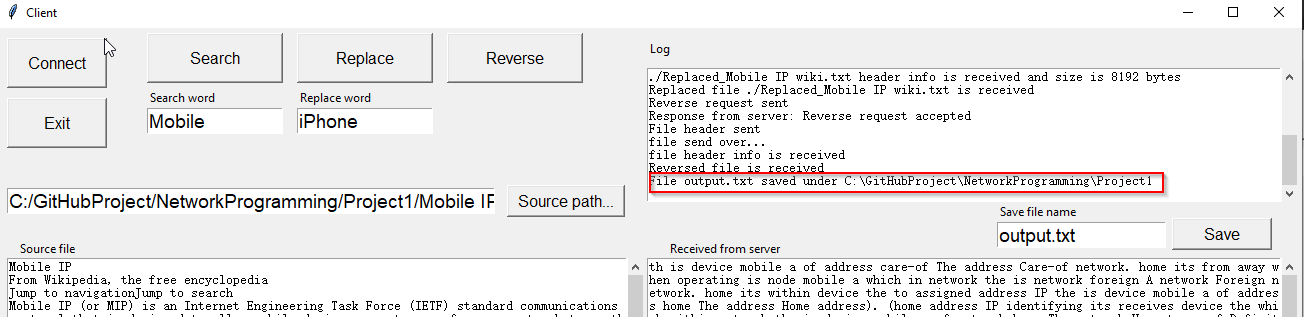
* + From the above figure, the word “client’ is replaced by “CLIENT”. The replaced file is received from server and showed in client program.
  + The info in Server program matches with Client’s
  + Log info is also showing detail steps of the communication.

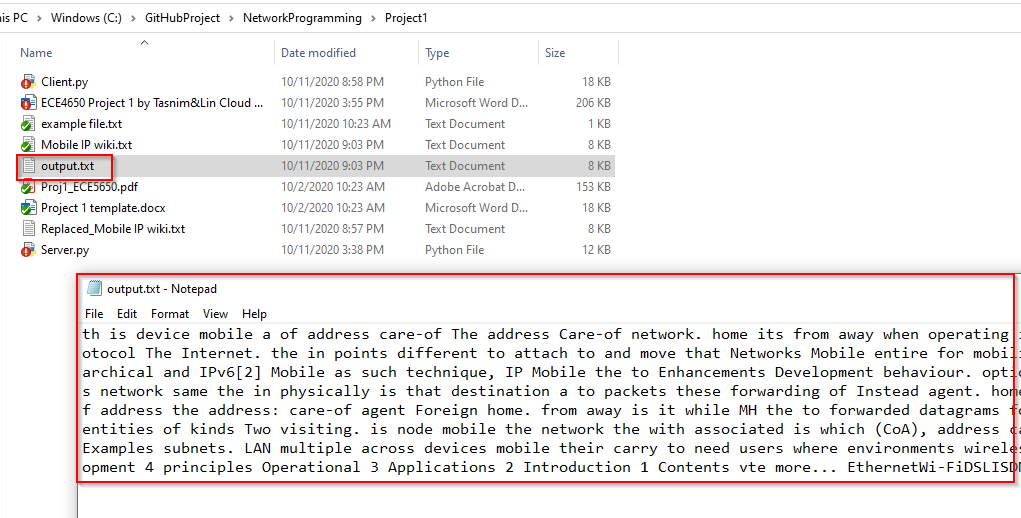
1. **Reverse function**

****

* + Server received the source file sent from client. From the content showing above , it is same as the source file.
  + Client received the reversed file sent from Server. From the content showing above, it is same as the processed content in server.
  + Log info is also showing detail steps of the communication.

1. **Save file**

****

****

**Completion Status and Self-Critique:**

For program **Client.py**:

* Does your program meet all requirements? If not, explain the problem.

Yes

* Does the program run correctly all the time? If not, explain the problem.

Yes

* Did you adequately test the program? If not, specify.

Yes

* Is the program well documented?

Yes

For program **Server.py**:

* Does your program meet all requirements? If not, explain the problem.

Yes

* Does the program run correctly all the time? If not, explain the problem.

Yes

* Did you adequately test the program? If not, specify.

Yes

* Is the program well documented?

Yes