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

**Project 2**

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

**Source Code(s):**

**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

**import** tkinter**.**scrolledtext **as** ScrolledText

**import** os

**import** sys

**import** struct

**import** threading

**import** win32api

**from** datetime **import** datetime

**import** time

**import** zipfile

**import** csv

**import** lzma

serverName **=** '192.168.0.15' #'192.168.0.15' 'localhost'

serverPort **=** 12000

appName **=** 'Client01'

zipType **=** 'lzma' #'lzma' 'zipfile'

CurrentDirectory **=** os**.**getcwd**()**

#Create main GUI window

window **=** tk**.**Tk**()**

#Set widnow's title

window**.**title**(**appName**)**

#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**(**'Alice'**)**

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**(**appName**)**

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/Project2/{}/alice.txt'**.**format**(**appName**))**

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'**)**

#Add check button

IsCompressedVar **=** tk**.**IntVar**()**

checkButton **=** tk**.**Checkbutton**(**window**,** text**=**'File compression enable'**,**variable**=**IsCompressedVar**,** onvalue**=**1**,** offvalue**=**0**)**

checkButton**.**place**(**x**=**150**,** y**=**120**,** 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)

#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** RequestThread**(**ReqType**,**ReqMsg**):**

**global** isConnected

record **=** **[]**

**if** isConnected**:**

startTime **=** datetime**.**now**()**

record**.**append**(**'\'' **+** str**(**startTime**))**

record**.**append**(**ReqType**)**

**try:**

#Send request to server

clientSocket**.**send**(**ReqMsg**.**encode**())**

**except** socket**.**error **as** msg**:**

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

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

isConnected **=** **False**

**else:**

LoggingText**.**insert**(**'insert'**,**'{} request sent\n'**.**format**(**ReqType**))**

#Receive message from server

response **=** clientSocket**.**recv**(**1024**)**

**if** response**:**

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

expectedResponse **=** '{} request accepted'**.**format**(**ReqType**)**

**if** response**.**decode**()** **==** expectedResponse**:**

filepath **=** SourceFilePathVar**.**get**()**

filename **=** os**.**path**.**basename**(**filepath**)**

filesize **=** os**.**stat**(**filepath**).**st\_size

record**.**append**(**filesize**)**

**if** os**.**path**.**isfile**(**filepath**):**

**if** IsCompressedVar**.**get**()** **==** 1**:**

zipStartTime **=** datetime**.**now**()**

**if** zipType **==**'zipfile'**:**

#Zipfile compression

zipfilename **=** filename**.**split**(**'.'**)[**0**]** **+** '.zip'

**with** zipfile**.**ZipFile**(**zipfilename**,** 'w'**,** zipfile**.**ZIP\_DEFLATED**)** **as** f**:**

f**.**write**(**filename**)**

**elif** zipType **==**'lzma'**:**

#lzma compression

zipfilename **=** filename**.**split**(**'.'**)[**0**]** **+** '.xz'

**with** lzma**.**open**(**zipfilename**,** 'wb'**)** **as** f**:**

**with** open**(**filename**,**'rb'**)** **as** pf**:**

textContent **=** pf**.**read**()**

f**.**write**(**textContent**)**

**else:**

**print(**'Wrong ziptype set!'**)**

filepath **=** zipfilename

filename **=** zipfilename

filesize **=** os**.**stat**(**filepath**).**st\_size

record**.**append**(**filesize**)**

zipDuration **=** datetime**.**now**()-**zipStartTime

record**.**append**(**'\'' **+** str**(**zipDuration**))**

**else:**

record**.**append**(**'None'**)**

record**.**append**(**0**)**

#Send file info to server

#Header structure : file name lentgh = 128 bytes; filesize = 8bytes; IsCompressed = 4bytes(int)

fhead **=** struct**.**pack**(**'128sQI'**,** bytes**(**filename**.**encode**(**'utf-8'**)),** filesize**,** IsCompressedVar**.**get**())**

clientSocket**.**send**(**fhead**)**

LoggingText**.**insert**(**'insert'**,** '{} file header sent\n'**.**format**(**ReqType**))**

sendStartTime **=** datetime**.**now**()**

#Send data to server

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

data **=** fp**.**read**()**

clientSocket**.**sendall**(**data**)**

LoggingText**.**insert**(**'insert'**,** '{} file send over...\n'**.**format**(**ReqType**))**

sendoverTime **=** datetime**.**now**()**

sendoverDetal **=** sendoverTime **-** sendStartTime

record**.**append**(**'\'' **+** str**(**sendoverDetal**))**

LoggingText**.**insert**(**'insert'**,** 'Waiting for server processing and feedback\n'**)**

rcvStartTime **=** datetime**.**now**()**

#4. Receive the processed result

fileinfo\_size **=** struct**.**calcsize**(**'128sQI'**)**

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

**if** fileinfo\_data**:**

filename**,**filesize**,**IsCompressed **=** struct**.**unpack**(**'128sQI'**,**fileinfo\_data**)**

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

LoggingText**.**insert**(**'insert'**,** 'Processed file header info is received for {}\n'**.**format**(**ReqType**))**

received\_size **=** 0

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

**while** **not** **(**received\_size **==** filesize**):**

**if(**filesize **-** received\_size **>** 4096**):**

data **=** clientSocket**.**recv**(**4096**)**

**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**

rcv\_file\_handle**.**write**(**data**)**

LoggingText**.**insert**(**'insert'**,** 'Processed file for {} is received\n'**.**format**(**ReqType**))**

reverseoverDetal **=** datetime**.**now**()** **-** rcvStartTime

record**.**append**(**'\'' **+** str**(**reverseoverDetal**))**

**if** isConnected**:**

**if** IsCompressed**:**

LoggingText**.**insert**(**'insert'**,** 'Processed file for {} was compressed\n'**.**format**(**ReqType**))**

unzipStartTime **=** datetime**.**now**()**

**if** zipType **==**'zipfile'**:**

**with** zipfile**.**ZipFile**(**rcv\_file\_name**,** 'r'**)** **as** zf**:**

filepath **=** zf**.**extract**(**zf**.**namelist**()[**0**])** #suppose only one file

#rcv\_file\_name = os.path.basename(filepath)

rcv\_file\_name **=** filepath

**elif** zipType **==**'lzma'**:**

**with** lzma**.**open**(**rcv\_file\_name**,** 'rb'**)** **as** f**:**

zipContent **=** f**.**read**()**

localFileName **=** 'ReceivedProcessedFor{}.txt'**.**format**(**ReqType**)**

**with** open**(**localFileName**,**'w'**)** **as** uf**:**

uf**.**write**(**zipContent**.**decode**(**"utf-8"**))**

rcv\_file\_name **=** localFileName

#Dsiplay partial content in GUI

#ProcessedFileText.delete(1.0,'end')

#ProcessedFileText.insert('insert', zipContent.decode("utf-8")[0:1000])

unzipDuration **=** datetime**.**now**()** **-** unzipStartTime

record**.**append**(**'\'' **+** str**(**unzipDuration**))**

LoggingText**.**insert**(**'insert'**,** 'Processed file for {} is decompressed\n'**.**format**(**ReqType**))**

**else:**

record**.**append**(**0**)**

total\_duration **=** datetime**.**now**()** **-** startTime

record**.**append**(**'\'' **+** str**(**total\_duration**))**

**with** open**(**rcv\_file\_name**,**'rb'**)** **as** rf**:**

all\_data\_str **=** rf**.**read**().**decode**(**'utf-8'**)**

#LoggingText.insert('insert', 'Processed file is stored locally\n')

#Dsiplay partial content in GUI

ProcessedFileText**.**delete**(**1.0**,**'end'**)**

ProcessedFileText**.**insert**(**'insert'**,** all\_data\_str**[**0**:**2000**])**

**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'**)**

**print(**'Request thread for {} ended'**.**format**(**ReqType**))**

**with** open**(**'record.csv'**,**'a+'**)** **as** csv\_record**:**

csv\_write **=** csv**.**writer**(**csv\_record**)**

csv\_write**.**writerow**(**record**)**

**def** SearchWordFromServer**():**

ReqType **=** 'Search'

search\_word **=** SearchWordVar**.**get**()**

ReqMsg **=** 'SEARCH+' **+** search\_word

search\_thread **=** threading**.**Thread**(**target**=**RequestThread**,** name**=**'RequestThread'**,** args**=(**ReqType**,**ReqMsg**),** daemon**=True)**

#search\_thread.setDaemon(True)

search\_thread**.**start**()**

**print(**'RequestThread started for Search function'**)**

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**():**

ReqType **=** 'Replace'

search\_word **=** SearchWordVar**.**get**()**

replace\_word **=** ReplaceWordVar**.**get**()**

ReqMsg **=** 'REPLACE+' **+** search\_word **+**'+' **+** replace\_word

replace\_thread **=** threading**.**Thread**(**target**=**RequestThread**,** name**=**'RequestThread'**,** args**=(**ReqType**,**ReqMsg**),** daemon**=True)**

#replace\_thread.setDaemon(True)

replace\_thread**.**start**()**

**print(**'RequestThread started for Replace function'**)**

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**():**

ReqType **=** 'Reverse'

ReqMsg **=** 'REVERSE'

reverse\_thread **=** threading**.**Thread**(**target**=**RequestThread**,** name**=**'RequestThread'**,** args**=(**ReqType**,**ReqMsg**),** daemon**=True)**

#reverse\_thread.setDaemon(True)

reverse\_thread**.**start**()**

**print(**'RequestThread started for Reverse function'**)**

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**()**

#print(len(text))

SourceFileText**.**insert**(**'insert'**,**text**[**0**:**1000**])**

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**()**

**Source code for Server.py:**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# File Name：Server.py

# Created: 11/04/2020

# Author: Li Lin & Anika Tasnim

**import** os

**import** sys

**import** socket

**import** threading

**import** tkinter **as** tk

**import** tkinter**.**scrolledtext **as** ScrolledText

**from** tkinter **import** ttk

**from** datetime **import** datetime

**import** time

**import** struct

**import** re

**import** zipfile

**import** csv

**import** lzma

serverPort **=** 12000

zipType **=**'zipfile'

#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'**)**

client1Pbar **=** ttk**.**Progressbar**(**window**,** length**=**200**,**cursor**=**'spider'**,**

mode**=**"determinate"**,**

orient**=**tk**.**HORIZONTAL**)**

client1Pbar**.**place**(**x**=**800**,** y**=**70**,** anchor**=**'nw'**)**

client1Pbar**[**'value'**]** **=** 0

client2Pbar **=** ttk**.**Progressbar**(**window**,** length**=**200**,**cursor**=**'spider'**,**

mode**=**"determinate"**,**

orient**=**tk**.**HORIZONTAL**)**

client2Pbar**.**place**(**x**=**800**,** y**=**100**,** anchor**=**'nw'**)**

client2Pbar**[**'value'**]** **=** 0

#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'**)**

Client1StatusLabel **=** tk**.**Label**(**window**,** text**=**'Client1'**)**

Client1StatusLabel**.**place**(**x**=**750**,** y**=**70**,** anchor**=**'nw'**)**

Client2StatusLabel **=** tk**.**Label**(**window**,** text**=**'Client2'**)**

Client2StatusLabel**.**place**(**x**=**750**,** y**=**100**,** anchor**=**'nw'**)**

#Add check button

IsCompressedVar **=** tk**.**IntVar**()**

checkButton **=** tk**.**Checkbutton**(**window**,** text**=**'Send compressed file '**,**variable**=**IsCompressedVar**,** onvalue**=**1**,** offvalue**=**0**)**

checkButton**.**place**(**x**=**750**,** y**=**30**,** anchor**=**'nw'**)**

serverStarted **=** **False**

**def** StartServerThread**():**

**global** serverStarted

serverStarted **=** **True**

LoggingText**.**insert**(**'insert'**,** 'Server Started\n'**)**

ClinetId **=** 0

**try:**

**while** **True:**

#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**)**

#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 from {}:{}\n'.format(addr[0],addr[1]))

rcv\_thread = threading.Thread(target=ReceiveDataThread, name = 'Thread{}'.format(addr[1]), args=(connection,addr, ClinetId), daemon=True)

rcv\_thread.start()

ClinetId = ClinetId + 1

except socket.error as msg:

print(msg)

LoggingText.insert('insert', 'Server error:{}. Server closed\n'.format(msg))

else:

print("Server closed")

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

serverStarted = False

def UpdateLoggingToEnd():

while True:

LoggingText.see('end')

time.sleep(0.5)

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

#UpdateLogging\_thread.start()

def StartServer():

global serverStarted

if serverStarted == False:

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

start\_thread.start()

else:

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

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, clientId):

isConnected = True

request = ''

search\_word = ''

replace\_word = ''

clientIp = address[0]

clientPort = address[1]

foldername = str(clientIp)+'\_'+str(clientPort)

isExisting = os.path.exists(foldername)

if not isExisting:

os.makedirs(foldername)

recordFilename = os.path.join('./{}/'.format(foldername), foldername + "record.csv")

with open(recordFilename,'w') as csv\_record:

csv\_write = csv.writer(csv\_record)

csv\_head = ['Client','StartTime','RequestType','ReceivedDuration','RcvFileSize','UnzipDuration','ReadDuration','ProcessDuration','StoreDuration','ZipDuration','ReplyDuration','ReplyFileSize','TotalDuration']

csv\_write.writerow(csv\_head)

with connection:

status = 'WAIT\_FOR\_REQUEST'

while isConnected:

print('waiting receive data')

LoggingText.insert('insert', '{}:{} waiting receive data\n'.format(clientIp,clientPort))

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

#Receive message from client

message = connection.recv(1024)

if not message:

isConnected = False

break

#Set progress bar

# if clientId == 0:

# client1Pbar['value'] = 5

# else:

# client2Pbar['value'] = 5

message = message.decode()

#print(message)

record = [foldername]

startTime = datetime.now()

record.append('\'' + str(startTime))

reqType = ''

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

reqType = 'Search'

logMsg = '{}:{} Search request received and accepted\n'.format(clientIp,clientPort)

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:

reqType = 'Replace'

logMsg = '{}:{} Replace request received and accepted\n'.format(clientIp,clientPort)

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':

reqType = 'Reverse'

logMsg = '{}:{} Reverse request received and accepted\n'.format(clientIp,clientPort)

request = 'REVERSE'

message = 'Reverse request accepted'

connection.send(message.encode())

status = 'WAIT\_FOR\_FILE\_INFO'

elif message == 'EXIT':

reqType = 'Exit'

logMsg = '{}:{} Exit request received\n'.format(clientIp,clientPort)

connection.close()

isConnected = False

else:

reqType = 'Unknown'

message ='unrecognized request!'

LoggingText.insert('insert', logMsg)

record.append(reqType)

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 7

else:

client2Pbar['value'] = 7

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

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

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

if not fileinfo\_data:

isConnected = False

break

#Receive file name and size info

filename,filesize,IsCompressed = struct.unpack('128sQI',fileinfo\_data) #file name lentgh = 128 bytes; filesize = 8bytes; I:unsigned int for compression

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

LoggingText.insert('insert', '{}:{} Header info received\n'.format(clientIp,clientPort))

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 10

else:

client2Pbar['value'] = 10

#Receive the data of file

received\_size = 0

rcvStartTime = datetime.now()

pre\_process\_file = os.path.join('./{}/'.format(foldername), rcv\_file\_name)

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

while not (received\_size == filesize):

if(filesize - received\_size > 4096):

data = connection.recv(4096)

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', '{}:{} Received {} bytes data\n'.format(clientIp, clientPort,received\_size))

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 10 + int((received\_size/filesize)\*40)

else:

client2Pbar['value'] = 10 + int((received\_size/filesize)\*40)

LoggingText.insert('insert', '{}:{} Requested file is received\n'.format(clientIp, clientPort))

rcvEndTime = datetime.now()

rcvDelta = rcvEndTime - rcvStartTime

record.append('\'' + str(rcvDelta))

record.append(filesize)

#record.append('\'' + str(rcvEndTime))

if isConnected == False:

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

else:

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

#Check if the received file is compressed

if IsCompressed:

LoggingText.insert('insert', '{}:{} Requested file was compressed\n'.format(clientIp, clientPort))

upzipStartTime = datetime.now()

if zipType =='zipfile':

with zipfile.ZipFile(pre\_process\_file, 'r') as zf:

filepath = zf.extract( zf.namelist()[0], './{}/'.format(foldername)) #suppose only one file

pre\_process\_file = filepath

elif zipType =='lzma':

with lzma.open(pre\_process\_file, 'rb') as f:

zipContent = f.read()

pre\_process\_file = 'ReceivedFileFor{}.txt'.format(request)

with open(pre\_process\_file,'w') as uf:

uf.write(zipContent.decode("utf-8"))

LoggingText.insert('insert', '{}:{} Requested file is decompressed\n'.format(clientIp, clientPort))

upzipDelta = datetime.now()-upzipStartTime

record.append('\'' + str(upzipDelta))

#record.append('\'' + str(datetime.now()))

else:

record.append('None')

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 55

else:

client2Pbar['value'] = 55

readStartTime = datetime.now()

#Read out the data from file

all\_data\_str = 'No Data'

with open(pre\_process\_file,'rb') as rf:

all\_data\_str = rf.read().decode('utf-8')

readDelta = datetime.now()-readStartTime

record.append('\'' + str(readDelta))

#record.append('\'' + str(datetime.now()))

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 57

else:

client2Pbar['value'] = 57

#Write data to GUI ReceivedText field

#ReceivedText.delete(1.0,'end')

#ReceivedText.insert('insert',all\_data\_str[0:1000])

ReceivedText.insert('insert','{}:{} Received file for {}\n'.format(clientIp, clientPort, reqType))

LoggingText.insert('insert', '{}:{} Requested file for {} is processing. \n'.format(clientIp, clientPort,reqType))

processStartTime = datetime.now()

#Process data according to request

if request == 'SEARCH':

#Search

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

processed\_data = 'There are {} words "{}" found in {}.'.format(count,search\_word,rcv\_file\_name)

processed\_file\_name = os.path.join('./{}/'.format(foldername), 'SearchResult\_' + rcv\_file\_name)

elif request == 'REPLACE':

#Replace

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

processed\_file\_name = os.path.join('./{}/'.format(foldername), 'ReplaceResult\_' + rcv\_file\_name)

elif request == 'REVERSE':

#Reverse

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

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

processed\_file\_name = os.path.join('./{}/'.format(foldername), 'ReverseResult\_' + rcv\_file\_name)

else:

processed\_data = 'unknown request!'

processed\_file\_name = os.path.join('./{}/'.format(foldername), 'unknowRequest.txt')

processDuration = datetime.now()-processStartTime

record.append('\'' + str(processDuration))

#record.append('\'' + str(datetime.now()))

LoggingText.insert('insert', '{}:{} Processing for {} is done. \n'.format(clientIp, clientPort,reqType))

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 60

else:

client2Pbar['value'] = 60

#ProcessedText.delete(1.0,'end')

#ProcessedText.insert('insert',processed\_data[0:1000])

ProcessedText.insert('insert','{}:{} Processed file for {}\n'.format(clientIp, clientPort, reqType))

storeStartTime = datetime.now()

#Store local file

LoggingText.insert('insert', '{}:{} Store processed file to local. \n'.format(clientIp, clientPort))

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

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

storeDuration = datetime.now() - storeStartTime

record.append('\'' + str(storeDuration))

#record.append('\'' + str(datetime.now()))

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 65

else:

client2Pbar['value'] = 65

#Check if send compressed file

if IsCompressedVar.get() == 1:

LoggingText.insert('insert', '{}:{} Conducting compression for feedback\n'.format(clientIp, clientPort))

zipStartTime = datetime.now()

if zipType =='zipfile':

compressFileName = os.path.join('./{}/'.format(foldername),'Processed.zip')

with zipfile.ZipFile(compressFileName, 'w', zipfile.ZIP\_DEFLATED) as f:

f.write(processed\_file\_name)

elif zipType =='lzma':

compressFileName = os.path.join('./{}/'.format(foldername),'Processed.xz')

with lzma.open(compressFileName, 'wb') as f:

with open(processed\_file\_name,'rb') as pf:

textContent = pf.read()

f.write(textContent)

LoggingText.insert('insert', '{}:{} Compression for feedback finished\n'.format(clientIp, clientPort))

filepath = compressFileName

processed\_file\_name = compressFileName

filesize = os.stat(compressFileName).st\_size

zipDuration = datetime.now() - zipStartTime

record.append('\'' + str(zipDuration))

#record.append('\'' + str(datetime.now()))

else:

filesize = os.stat(processed\_file\_name).st\_size

record.append('None')

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 70

else:

client2Pbar['value'] = 70

replyStartTime = datetime.now()

LoggingText.insert('insert', '{}:{} Start sending feedback for {} request\n'.format(clientIp, clientPort,reqType))

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

#Send file info to client

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

#define file head info, including name and size

filename = os.path.basename(processed\_file\_name)

fhead = struct.pack('128sQI', bytes(filename.encode('utf-8')), filesize, IsCompressedVar.get())

connection.send(fhead)

LoggingText.insert('insert', '{}:{} Processed file header info for {} sent\n'.format(clientIp, clientPort, request))

#send file data to client

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

connection.sendall(send\_data)

LoggingText.insert('insert', '{}:{} Processed file for {} send over\n'.format(clientIp, clientPort, request))

replyDuration = datetime.now() - replyStartTime

record.append('\'' + str(replyDuration))

record.append(filesize)

#record.append('\'' + str(datetime.now()))

TotalDuration = datetime.now()-startTime

record.append('\'' + str(TotalDuration))

with open(recordFilename,'a+') as csv\_record:

csv\_write = csv.writer(csv\_record)

csv\_write.writerow(record)

#Set progress bar

if clientId == 0:

client1Pbar['value'] = 100

else:

client2Pbar['value'] = 100

status = 'WAIT\_FOR\_REQUEST'

else:

print('Server is in unknown status')

status = 'WAIT\_FOR\_REQUEST'

LoggingText.insert('insert', '{}：{} Connection closed\n'.format(clientIp,clientPort))

window.mainloop()

**Source code for Compression.py:**

**import** gzip

**import** lzma

**import** zipfile

**import** os

**from** datetime **import** datetime

#source file

benchmark **=** **{}**

benchmark**[**'OrgSize'**]** **=** os**.**path**.**getsize**(**'alice\_Large.txt'**)**

**with** open**(**'alice\_Large.txt'**,**'rb'**)** **as** f**:**

textContent **=** f**.**read**()**

#gzip compression

gzipPara **=** **[]**

gzipStartTime **=** datetime**.**now**()**

**with** gzip**.**open**(**'alice\_Large\_gzip.gz'**,** 'wb'**)** **as** f**:**

f**.**write**(**textContent**)**

gzipDuration **=** str**(**datetime**.**now**()** **-** gzipStartTime**)**

gzipSize **=** os**.**path**.**getsize**(**'alice\_Large\_gzip.gz'**)**

gzipRatio **=** '{:.2%}'**.**format**(**gzipSize**/**benchmark**[**'OrgSize'**])**

gzipPara**.**append**(**gzipDuration**)**

gzipPara**.**append**(**gzipSize**)**

gzipPara**.**append**(**gzipRatio**)**

#gzip decompression

gunzipStartTime **=** datetime**.**now**()**

**with** gzip**.**open**(**'alice\_Large\_gzip.gz'**,** 'rb'**)** **as** f**:**

zipContent **=** f**.**read**()**

**with** open**(**'alice\_Large\_gzip\_unzip.txt'**,**'w'**)** **as** uf**:**

uf**.**write**(**zipContent**.**decode**(**"utf-8"**))**

gunzipDuration **=** str**(**datetime**.**now**()** **-** gunzipStartTime**)**

gunzipSize **=** os**.**path**.**getsize**(**'alice\_Large\_gzip\_unzip.txt'**)**

gzipPara**.**append**(**gunzipDuration**)**

gzipPara**.**append**(**gunzipSize**)**

benchmark**[**'gzip'**]** **=** gzipPara

#lzma compression

lzmaPara **=** **[]**

lzmaStartTime **=** datetime**.**now**()**

**with** lzma**.**open**(**'alice\_Large\_lzma.xz'**,** 'wb'**)** **as** f**:**

f**.**write**(**textContent**)**

lzmaDuration **=** str**(**datetime**.**now**()** **-** lzmaStartTime**)**

lzmaSize **=** os**.**path**.**getsize**(**'alice\_Large\_lzma.xz'**)**

lzmaRatio **=** '{:.2%}'**.**format**(**lzmaSize**/**benchmark**[**'OrgSize'**])**

lzmaPara**.**append**(**lzmaDuration**)**

lzmaPara**.**append**(**lzmaSize**)**

lzmaPara**.**append**(**lzmaRatio**)**

#lzma decompression

lzmaunzipStartTime **=** datetime**.**now**()**

**with** lzma**.**open**(**'alice\_Large\_lzma.xz'**,** 'rb'**)** **as** f**:**

zipContent **=** f**.**read**()**

**with** open**(**'alice\_Large\_lzma\_unzip.txt'**,**'w'**)** **as** uf**:**

uf**.**write**(**zipContent**.**decode**(**"utf-8"**))**

lzmaunzipDuration **=** str**(**datetime**.**now**()** **-** lzmaunzipStartTime**)**

lzmaunzipSize **=** os**.**path**.**getsize**(**'alice\_Large\_lzma\_unzip.txt'**)**

lzmaPara**.**append**(**lzmaunzipDuration**)**

lzmaPara**.**append**(**lzmaunzipSize**)**

benchmark**[**'lzma'**]** **=** lzmaPara

#zipfile compression

zipfilePara **=** **[]**

zipfileStartTime **=** datetime**.**now**()**

**with** zipfile**.**ZipFile**(**'alice\_Large\_zipfile.zip'**,** 'w'**,** zipfile**.**ZIP\_DEFLATED**)** **as** f**:**

f**.**write**(**'alice\_Large.txt'**)**

zipfileDuration **=** str**(**datetime**.**now**()** **-** zipfileStartTime**)**

zipfileSize **=** os**.**path**.**getsize**(**'alice\_Large\_zipfile.zip'**)**

zipfileRatio **=** '{:.2%}'**.**format**(**zipfileSize**/**benchmark**[**'OrgSize'**])**

zipfilePara**.**append**(**zipfileDuration**)**

zipfilePara**.**append**(**zipfileSize**)**

zipfilePara**.**append**(**zipfileRatio**)**

#zipfile decompression

zipfileunzipStartTime **=** datetime**.**now**()**

**with** zipfile**.**ZipFile**(**'alice\_Large\_zipfile.zip'**,** 'r'**)** **as** zf**:**

filepath **=** zf**.**extract**(**zf**.**namelist**()[**0**],** './zipfile/'**)** #suppose only one file

zipfileunzipDuration **=** str**(**datetime**.**now**()** **-** zipfileunzipStartTime**)**

zipfileunzipSize **=** os**.**path**.**getsize**(**'./zipfile/alice\_Large.txt'**)**

zipfilePara**.**append**(**zipfileunzipDuration**)**

zipfilePara**.**append**(**zipfileunzipSize**)**

benchmark**[**'zipfile'**]** **=** zipfilePara

**print(**benchmark**)**

**Flowchart about client and server communication:**



**Multithreading concept**

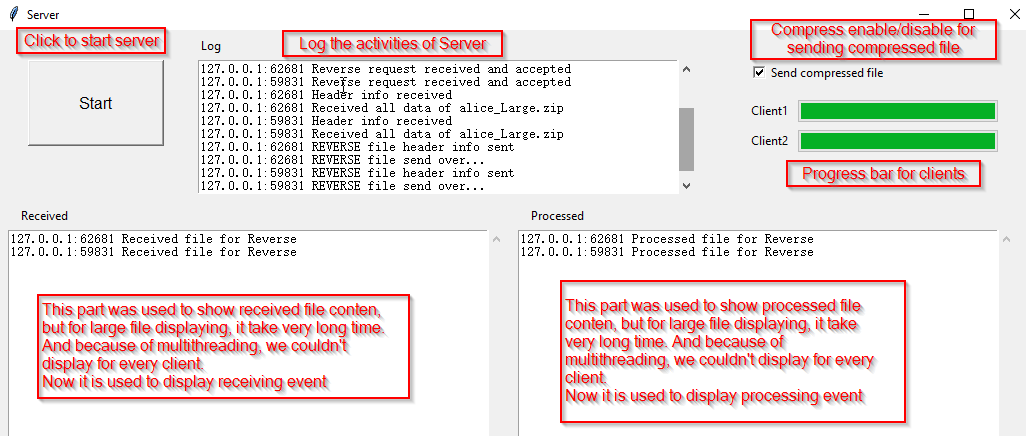


**Testing Procedure, including Description of Inputs:**

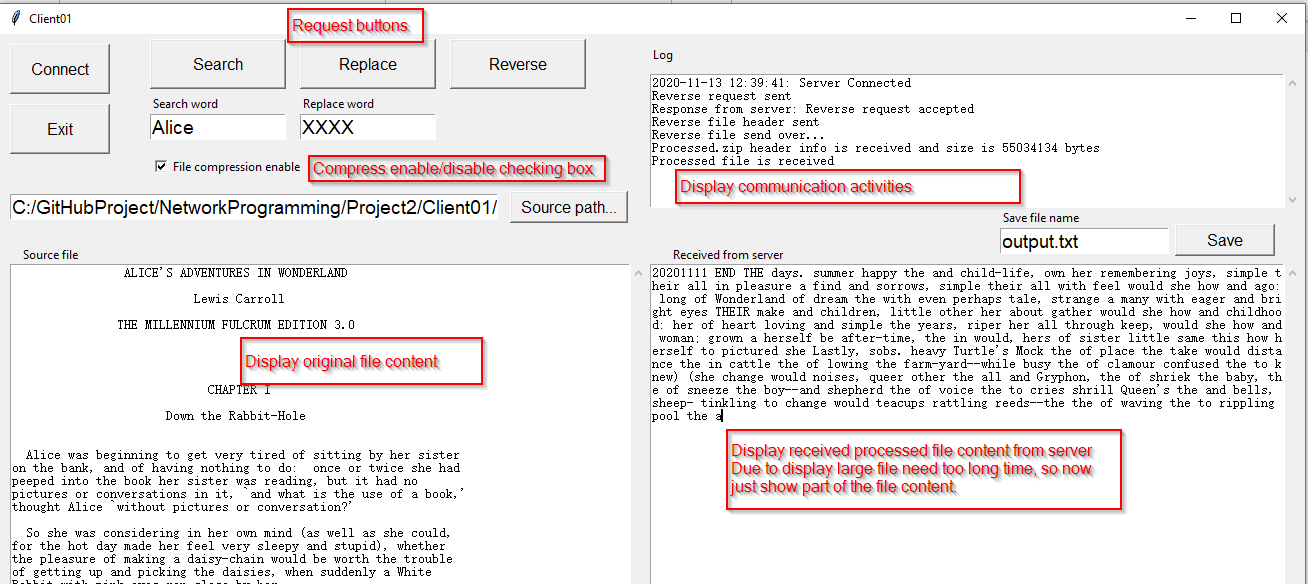
1. Test compression function in server and client program
   1. Run server and one client
   2. Enable compression flag on both server and client
   3. Make ‘Search’, ‘Replace’, and ‘Reverse’ request from client
   4. Enable compression flag on both server and client
   5. Make ‘Search’, ‘Replace’, and ‘Reverse’ request from client
   6. Check server should respond each request correctly
   7. Compare sending file time difference between before compression and after compression
2. Test if multithreading is working
   1. Run server program firstly
   2. Run client1 and client2 program(the two client code are same).
   3. Make ‘Search’, ‘Replace’, and ‘Reverse’ request from both clients
   4. Check server should respond each request correctly.
   5. Enable the compression flag on both server and client
   6. Make ‘Search’, ‘Replace’, and ‘Reverse’ request from both clients
   7. Check server should respond each request correctly.
3. Compare the performance

**Screenshots and Their Explanations:**

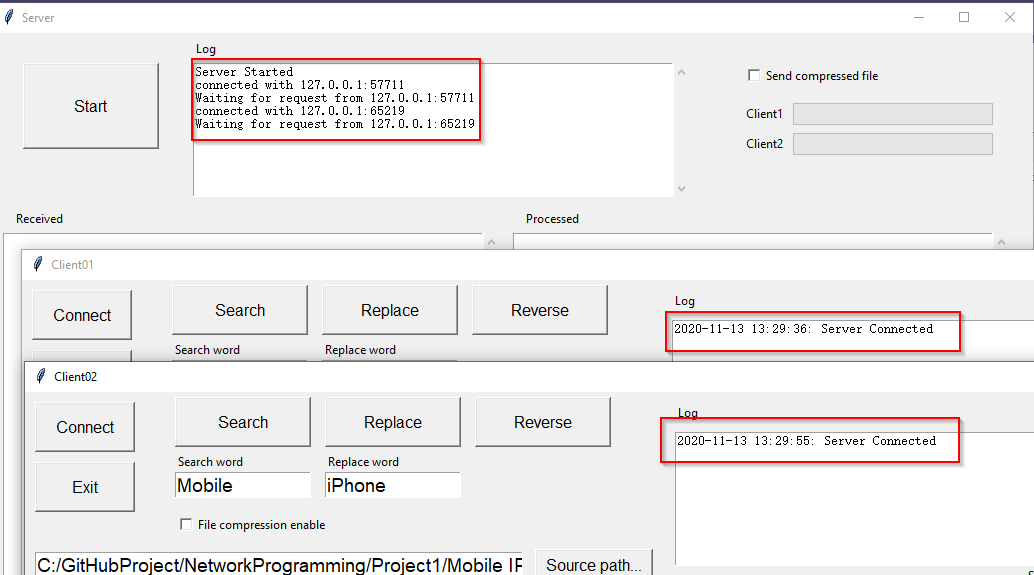
1. Server GUI

****

1. Client GUI

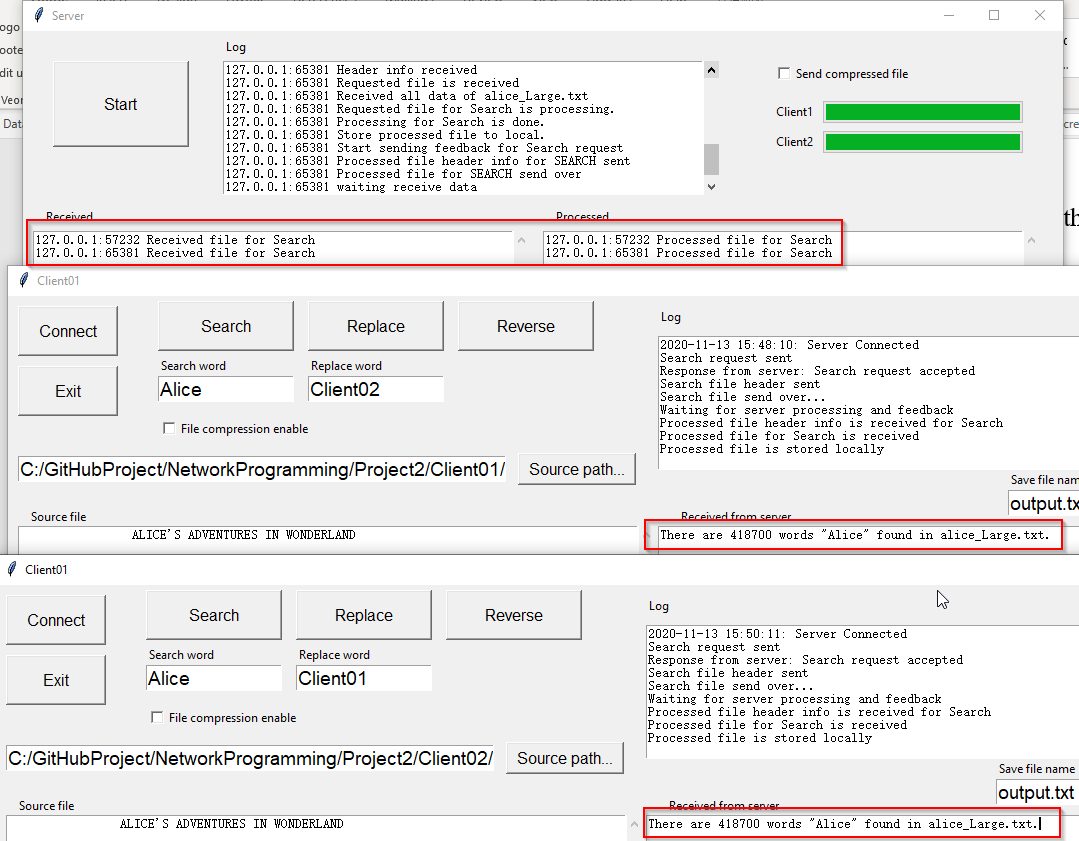


1. Confirm two client connecting with server at the same time

****

From above log info, we can see both client connect with server using new port numbers

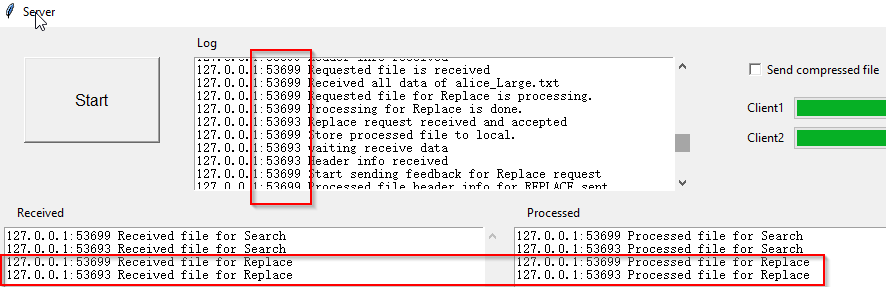
1. Check basic function ‘Search’ without compression



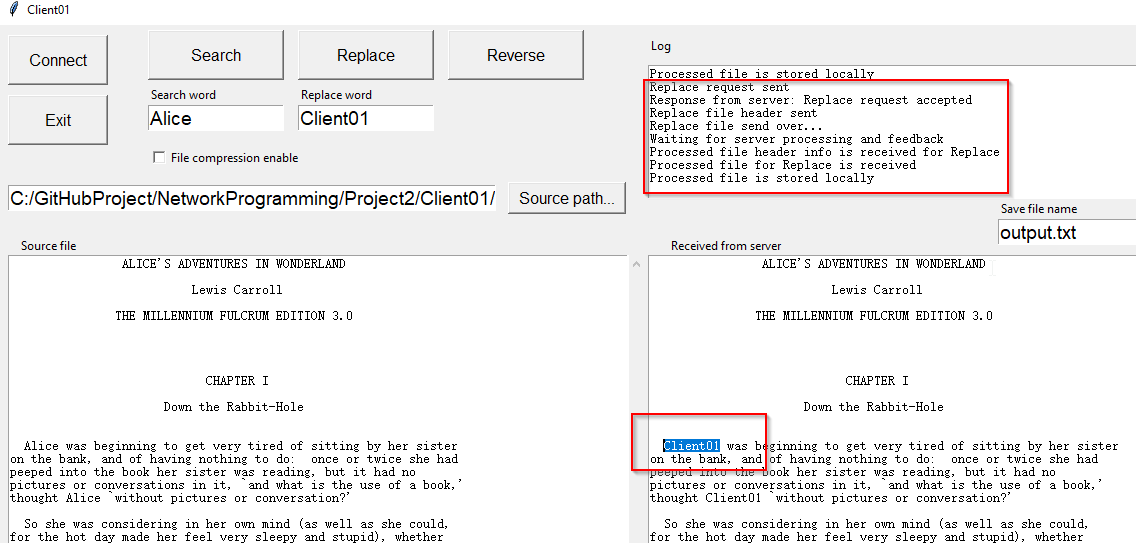
Both requests are accepted by server and file are processed, two clients received correct reponse.

1. Check basic function ‘Replace’ without compression

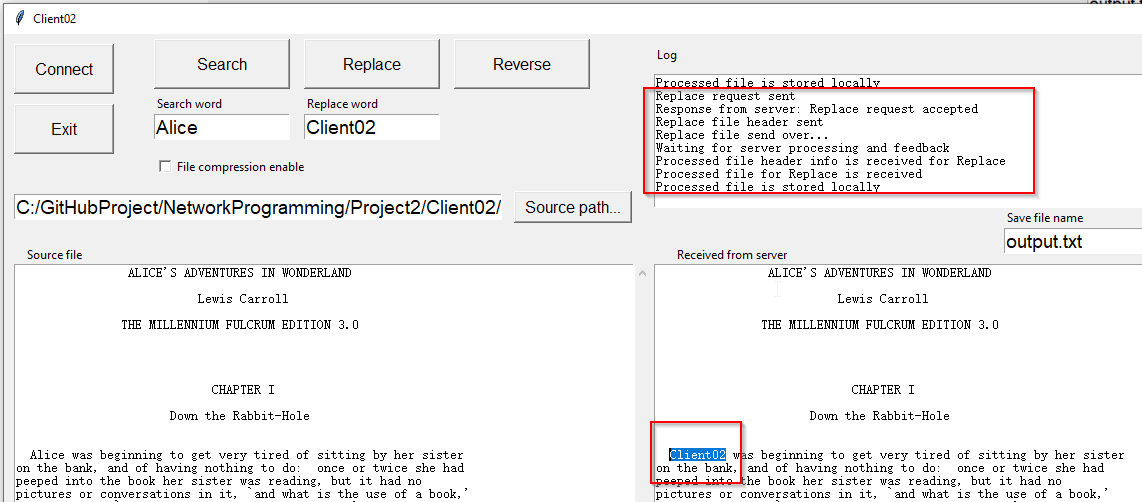
Server screen shows both request are processing at the same period



Client01 screen shows the replacement processing result is correct.

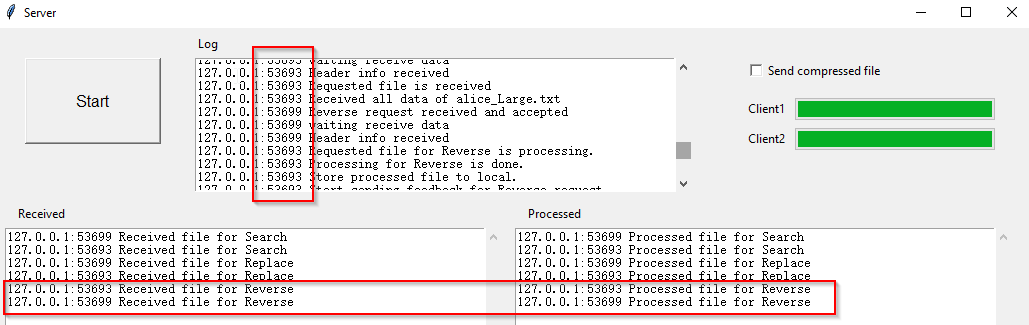


Client02 screen shows the replacement processing result is correct

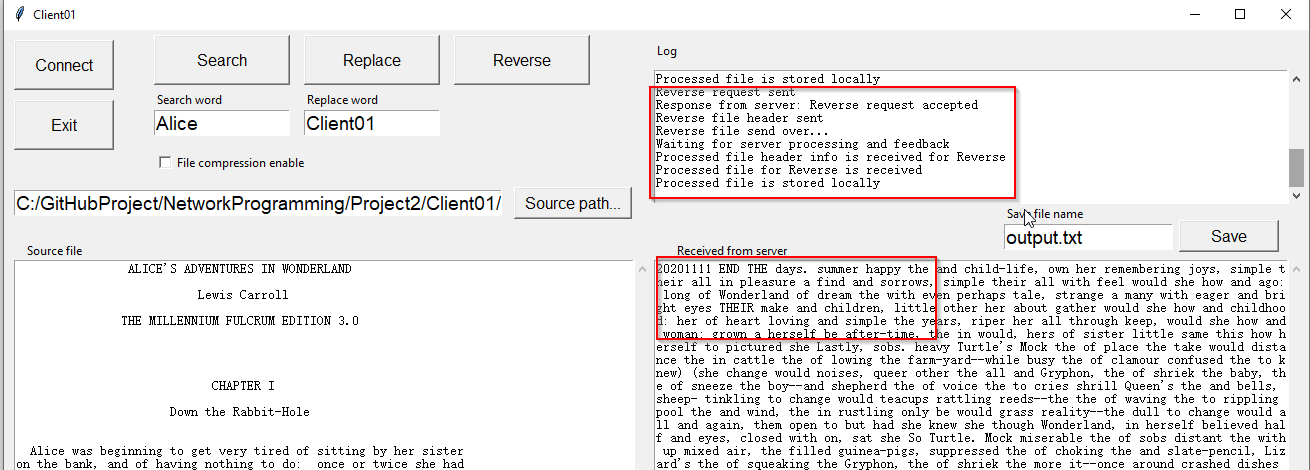


1. Check basic function ‘Reverse’ without compression

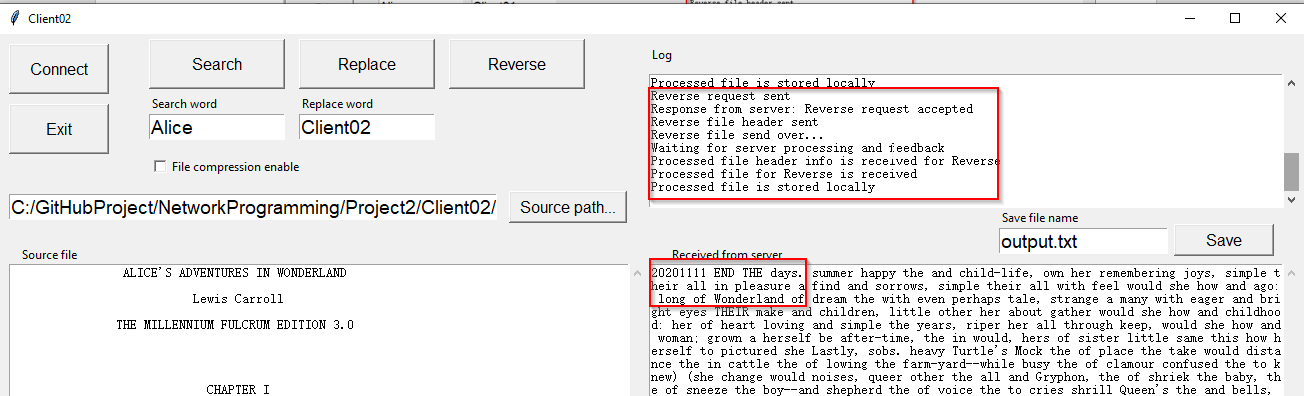
Server screen shows both request are processing at the same period



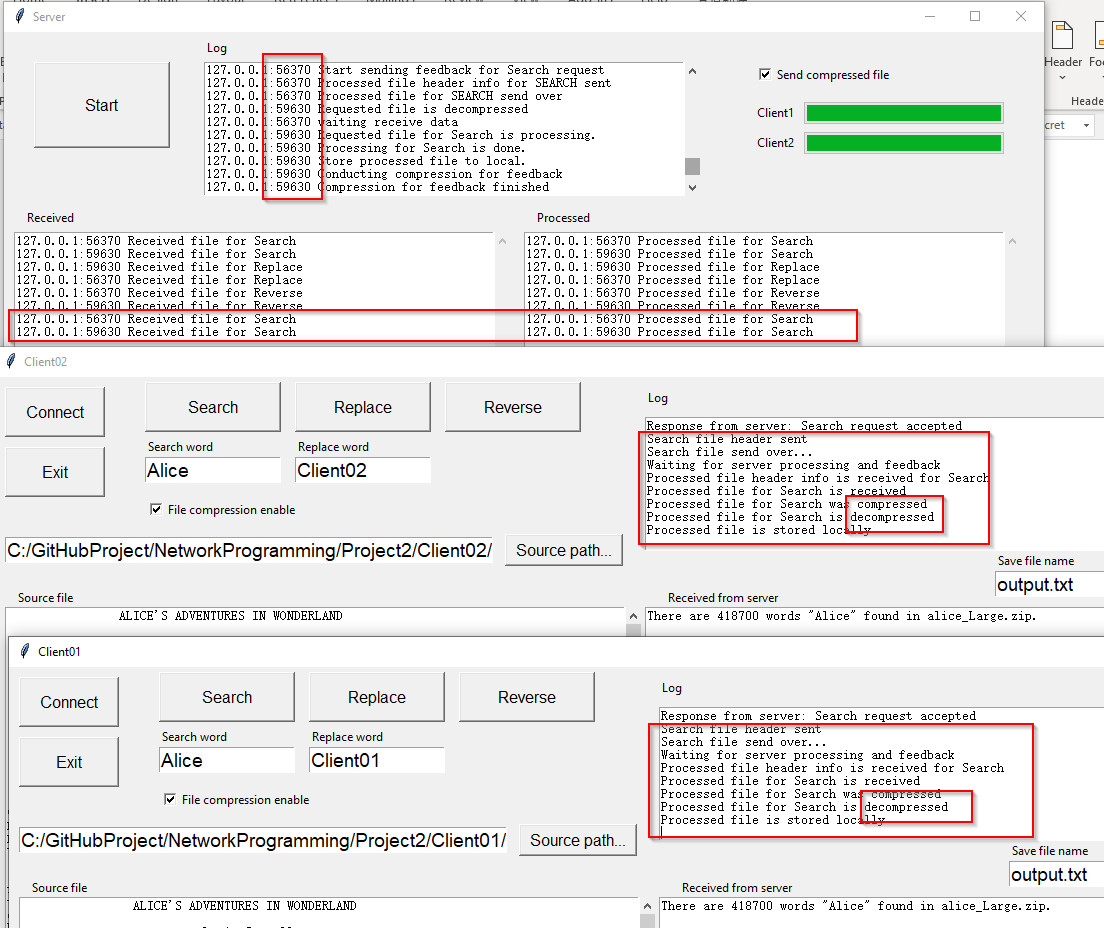
Client01 screen shows reverse process result is correct



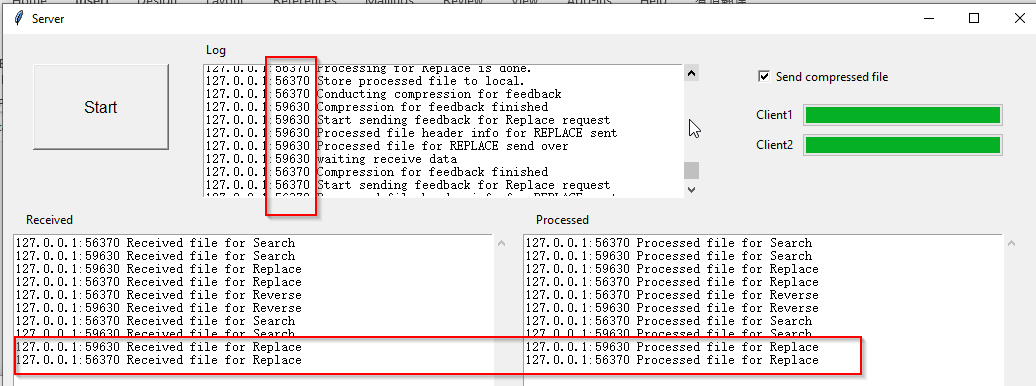
Client01 screen shows reverse process result is correct

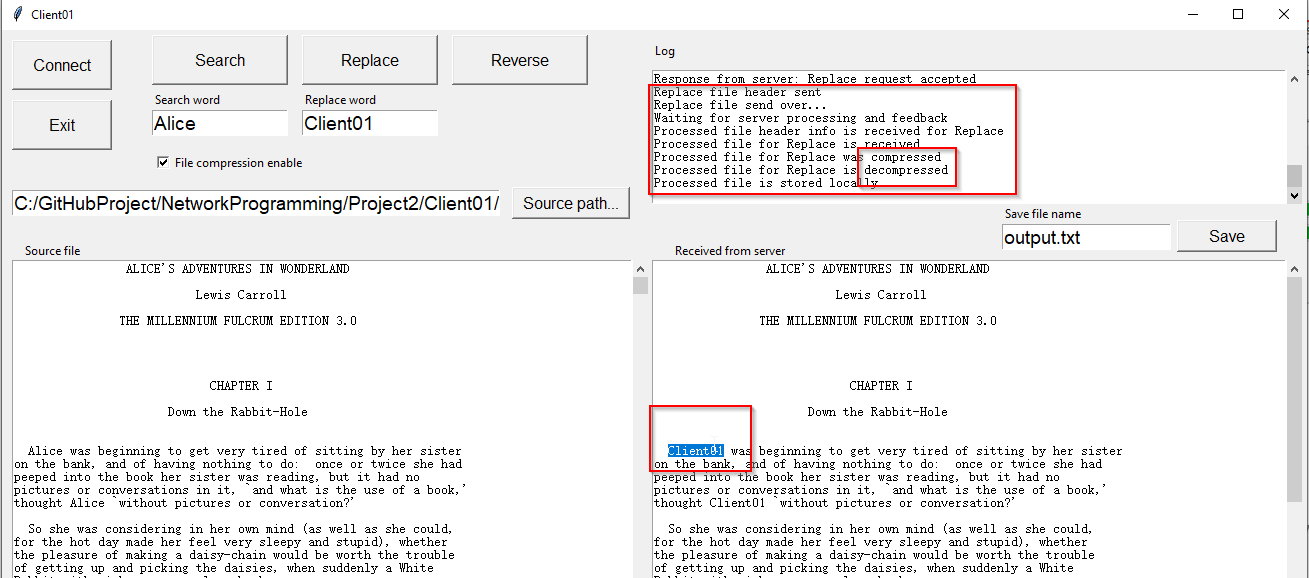


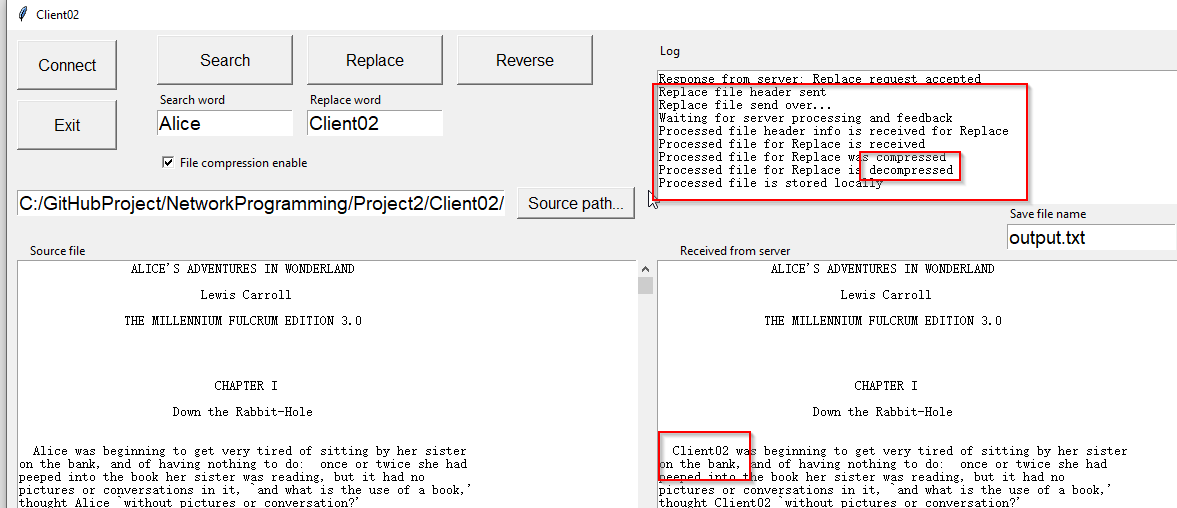
1. Check basic function ‘Search with compression



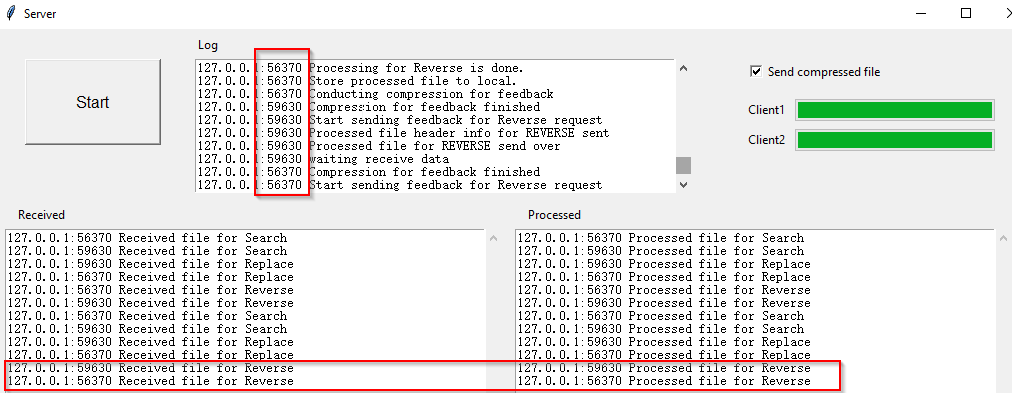
1. Check basic function ‘Replace’ with compression

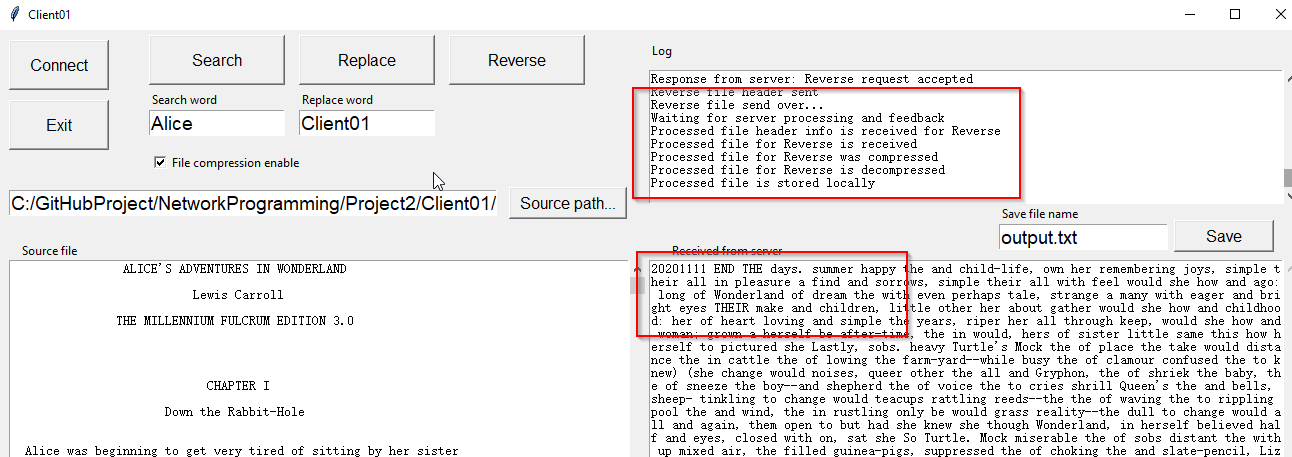


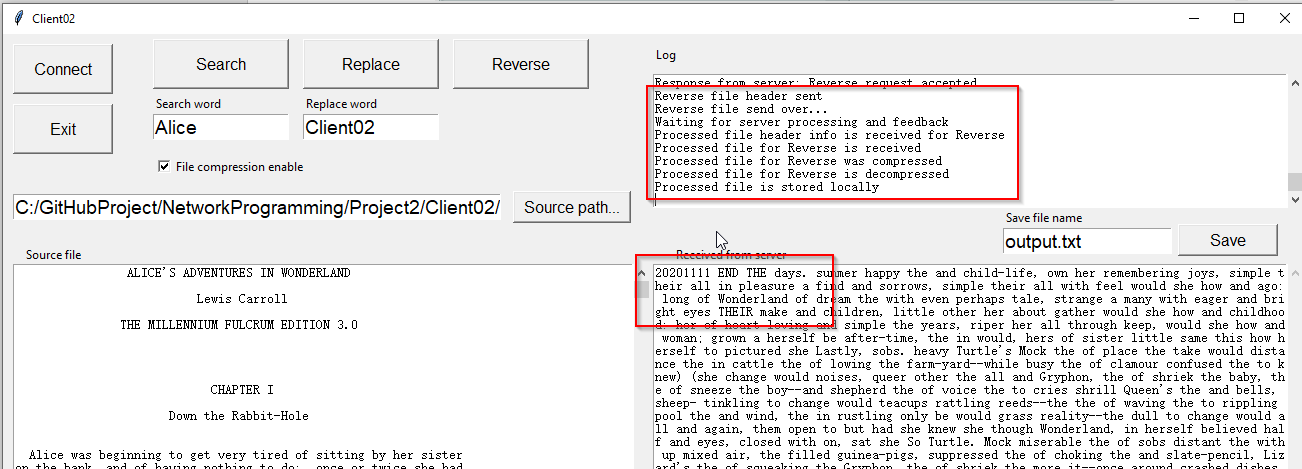




1. Check basic function ‘Reverse with compression







**Program 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

**Performance Evaluation Methodology:**

1. Benchmark for compression method(library) choosing

Check compression rate and time, and decompression time.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Compression benchmark** | | | | | | |
|  | Original File Size | Compress time | Compressed file size | Compress Ratio | Decompress time | Decompressed file size |
| **‘gzip'** Method | 161266323 | 17.51274 | 56129874 | '34.81%' | 02.584347' | 165080202 |
| **‘lzma'** Method | 161266323 | 46.50335 | 72056 | '0.04%' | 01.951976' | 165080202 |
| **‘zipfile'** Method | 161266323 | 11.88789 | 56434885 | '34.99%' | 01.136871' | 161266323 |

From above test result, lzma library has highest compression rate, but it spends longest compression time. Zipfile has shortest compression and decompression time, also the decompression file size is same as original, it means it is more accurate recovery.

1. **About server performance, one is about data processing efficiency and another is data transmission efficiency.**

**Data processing efficiency:** To improve data processing efficiency, we can use multi-threading. Each thread can be used for one client. This can be definitely improved for data processing efficiency. But the maximum number of thread is limited until full server CPU utilization.

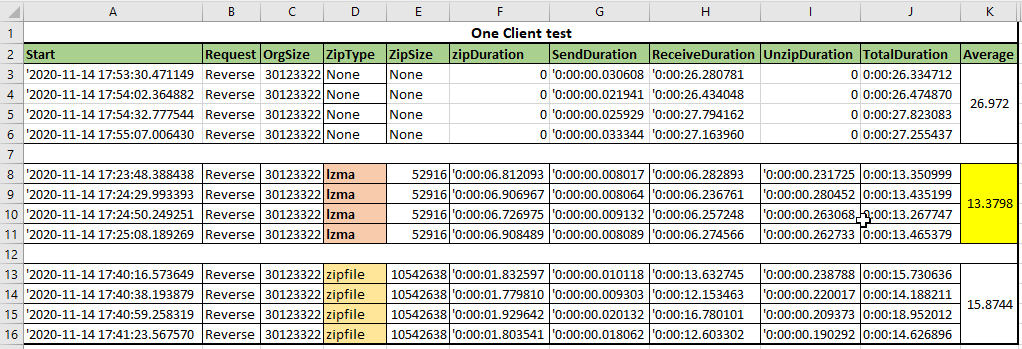
**Data transmission efficiency:** To improve data transmission efficiency, we can reduce the data size via compression. If the size is smaller , we do have shorter transmission time.

We need to consider the balance of compression time and communication time. T**he total time of one file processing is final condition we need to consider.**

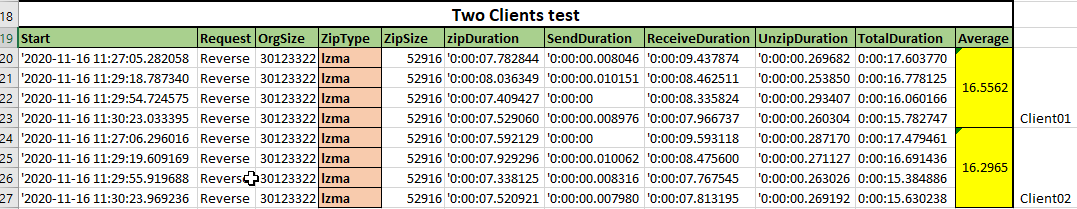
**Comparative Results:**

In my test, I use two computer in same local network. And in the code, I recorded the each step duration and total time into csv file.

1. Single client test

****

1. Two clients test(Multi-Threading)

****

**Analysis of the Results:**

1. From benchmark test for compression, we know gzip is a little bit worse than zipfile, so we don’t need to test for it.
2. From lamz and zipfile test result, we can know lamz is faster than zipfile under same size file.
3. From two clients test using multi-threading, server could process two same file around 16.5 seconds which means ,one file per 8.3 second faster than single client test 13.4 seconds. We also could increase the clients number to do the test, but conclusion is same: Multi-threading is more efficiency than single thread.