Jaime Perdue 2017843374 Project 4

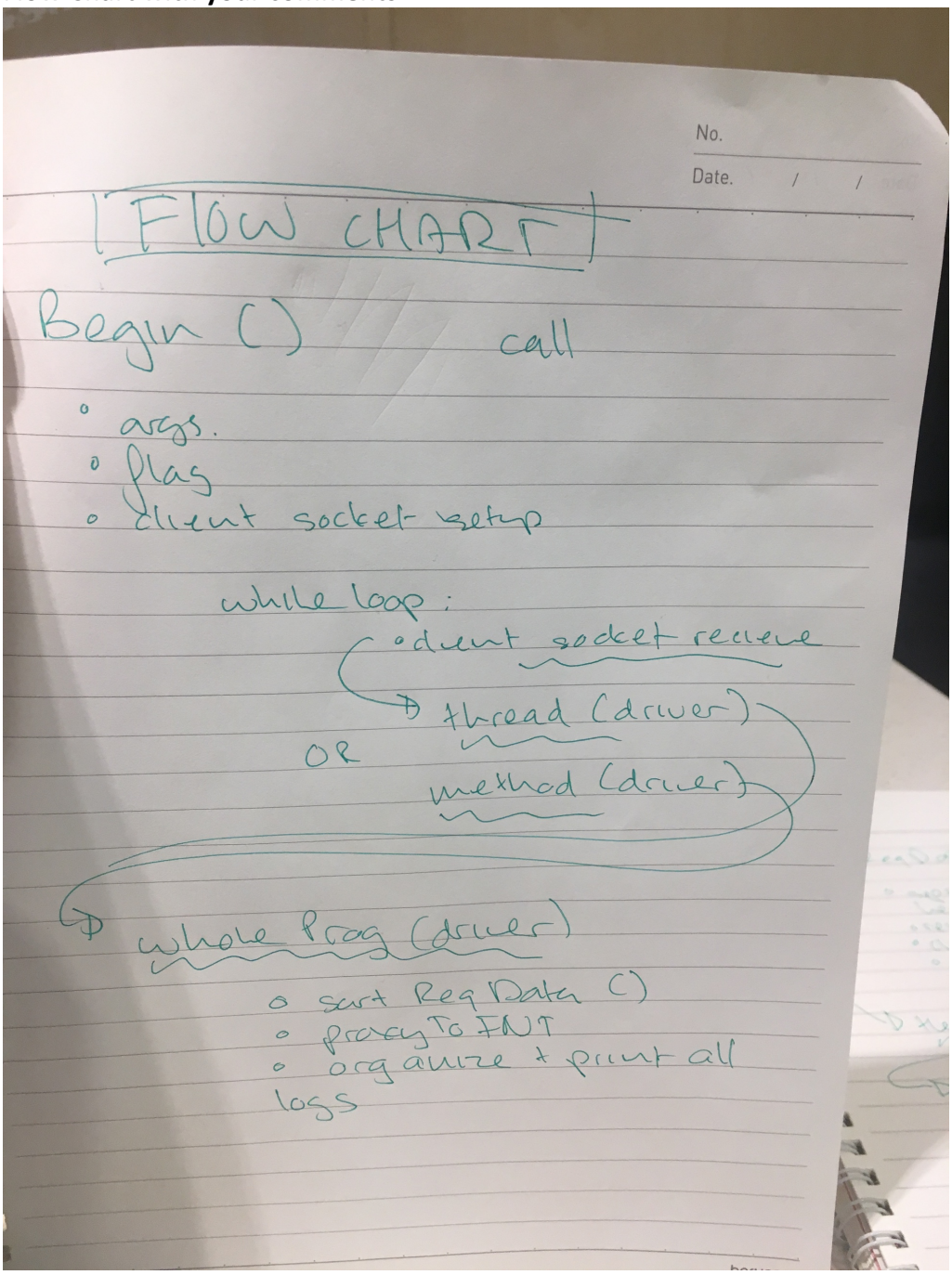
Explain how it works in detail:
Introduction:

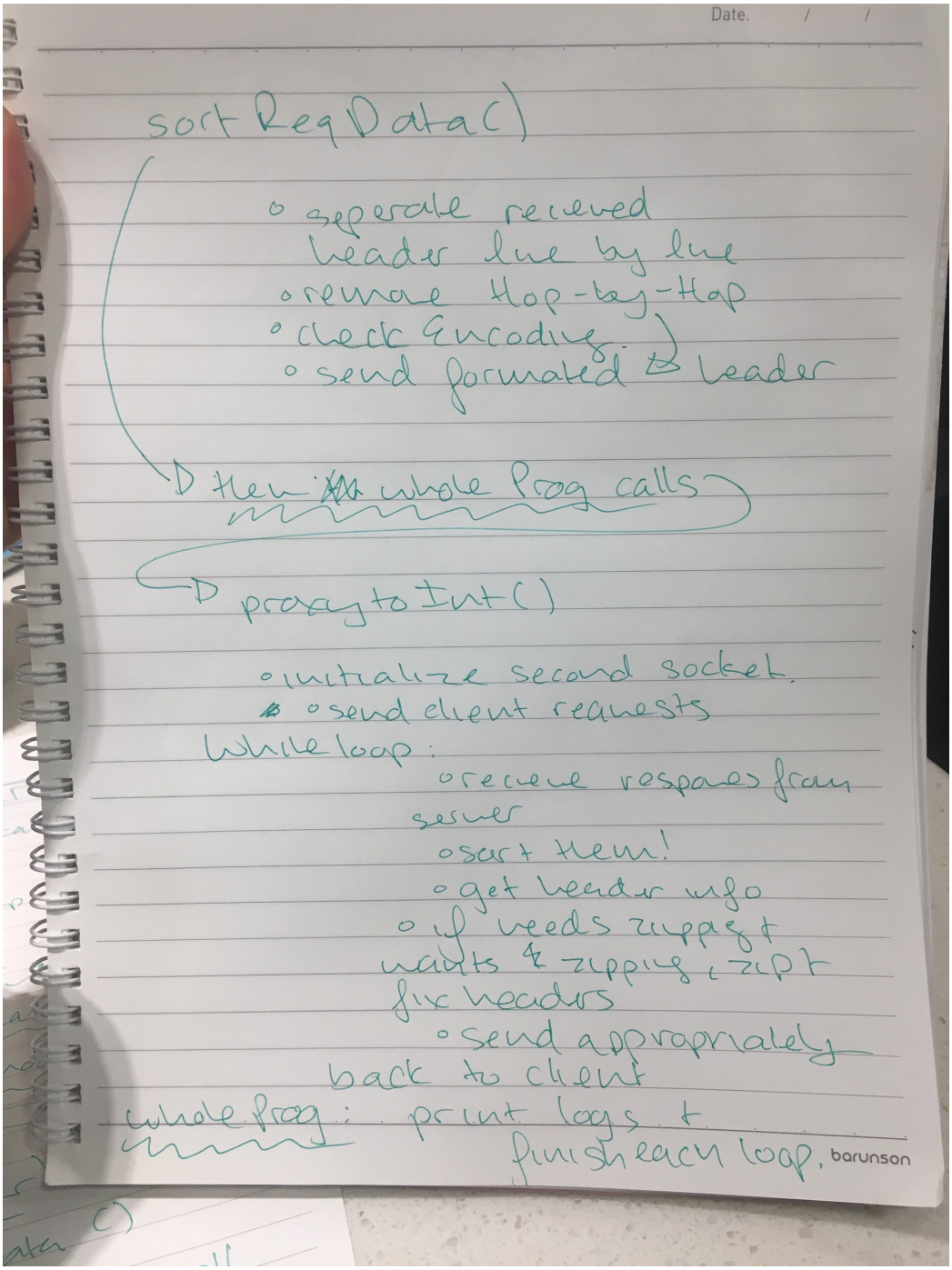
The coding language used is Python Version 2.7 on Ubuntu from Virtual Machine Virtual-Box. I imported argparse, sys, os, io, socket, stringIO and gzip thread and one threading call. I have three methods, begin, which calls the thread which calls a function sortReqData that sorts the data and sends it to the function proxToInt which sends it to the server. The Proxy then sends the item requested by the client and received by the server back to the client. My thread then prints the requested log. For the current project, my proxy now implements multithreading and payload compression, specifically with gzip files. When the user wants to implement either of these features they need to only set the flags that they'd like and specify the port. These command line arguments can be specified in either case and any order when executing the run script. They are then sent to the python file and used to turn those services on (via Booleans within the argument loop conditionals).
With multithreading, the Begin method calls the new multithread that uses the method WholeProg. This method works as a driver and takes in the client information. It then runs two methods to sort the information and then forward the sorted information to the server. The thread-method then logs the output for each cycle.
With the addition of compression and the compression flag. The program first makes sure that the client is able to receive zipped files by checking their header for the 'Accept-Encoding: gzip' header. If it does not contain that header we do not send them back a zip file. If they do have the correct header, we then check that the server sent back a zip file. We do this by sorting through the header of the response (only the first part of the very first request) and look line by line for "Content-Encoding: gzip". If they have that header, we can send simply send all the zipped data through the socket to the client as requested. If they do not have that header first insert the correct "Content-Encoding: gzip" header, and then we zip the data ourselves. We do so by separating the body after that first reply header and all of the subsequent responses and zipping them with gzip and StringIO. Then we send the headers and the zip file back to the client.
Describe why this makes your proxy server faster in detail.
Thanks to multithreading, the processor can simultaneously execute multiple threads at the same time. Because they are not subsequently waiting on each other, if one process is slow and holding up the rest, the others will not also be slowed down as much. Compression simply makes the data smaller as it transfers, and therefor takes less time and is faster.
Compare the performance(speed) before and after in detail
When neither flag is set the program is decidedly quite slow. 30 sec first req. 1min resp, 2 min another req, 3 min mostly rendered. **4 min fully rendered.**
With just compression on, a coupke of both requests and fully logged responses have gone through. By 2 minutes it has begun to load and finishes by **3 min 30 seconds or so**.

With just multithreading on 30 seconds gives several gets and 1 minute gives several logs. At almost exactly **2 minutes** its fully completed loading. It took exactly half the time as running without threading!
With **both** features on, immediately several requests and hellos, **within a minute** it has completely loaded!

Block(Function/Class) Diagram of your code
Begin() ->
Calles Begin which contains all the initial client and client socket establishing. It then enters a while loop ->
The while loop accepts the client data and creates a new thread with the client information. The Thread functions like a driver and calls all subsequent methods. ->
First it calls sortReqData. SortReqData not only removes unwanted headers but extracts specific pieces of information to be passed or later printed in the log.
Next the thread calls the proxy connection function proxyToInt which creates a new socket form the proxy to the server to send and receive the client's info. It also sorts the received information for printing.
If mt is on the it gets set in the beginning and called as a thread within that method. It then calls as subsequent methods as usual. The compression flag also progresses with the same flow and within the same methods but functions mostly within the proxyToInt method. The sortReq data is essential however, for making sure that the request header has accept encoding. Then that proxyToInt method and infuses the header and zips the files of those that are not already.

Flow Chart with your comments





Your careful comments of “block by block” with Code Screenshots of block by block.
In this first screenshots we see all of the imports and subsequently flags. We see header values being initialized to defaults and flags being set to off before they’re turned on by the user.

```
#!/usr/bin/python

import argparse
import sys
import os
import io
import StringIO
import gzip
from socket import *
from thread import *

#Section of flags and globak variable declarations
#These items are to be set and passed to various functions
#Many of then are used in the logging section in particular

#flags to implemented speed functions if requested
isProxyCompres = False
isThreading = False
#set to X by default and changed to O if user set flag
mul = 'X'
cp = 'X'
#header derived logging info
isOK = None
myme = None
sizu = 0
```

Here is a description of the function that is the core of the entire program. It calls all other funcions and is in charge of the logging. Each method goes back to this one until it all finishes.

```
#driver-like functioned called by multithread
#This function receives the client socket and information
#It then calls the method that extracts header info and sorts it
#it then calls the method that sends that info to the server, receives it
#and sends it back to the client
#it then prints the logs to the terminal

def wholeProg(clientConSock, clientAddr, count1, clientInfo):
#make sure the client isn't empty
        if clientInfo != "":
#pass correct info to formatting method
                host, port, formattedClientInfo = sortReqData(clientConSock, clientI
nfo, clientAddr)
#pass correct info to send to server method
                proxyToInt(host, port, formattedClientInfo, clientConSock, clientAdd
r)

#initialize empty string to append in order
#refer to global variables used in logs
#all logs
```

Here is simply the logging strings. They each take in either a passed or global variable derived either from the headers or from handmade set flags or counters

```
#initialize empty string to append in order
#refer to global variables used in logs
#all logs
                finalLog=""
                global mul
                global cp
                v1= ("starting proxy server on port " + str(clientAddr[1]))
                v18=("[" + mul + "]"  + " " + "MT"  + " " +  "|"  + " " + "["  + cp
 + "]"  + " " + "COMP"  + " " + "|" + " "  + "[" + "X" + "]" + " "  + "CHUNK" + " "
 +  "|" + " "  + "[" + "X" + "]" + " "  + "PC")
                v2= "----------------------------------------"
                v3= str(count1)
                v4=("[CLI connected to " + str(clientAddr[0]) + ":" + str(clientAddr
[1]) + "]")
                v5=("[CLI ==> PRX --- SRV]")
                v6=(" > " + str(typu) + " ")
                v7=("[CLI connected to " + str(host) + ":80" + "]")
                v8=("[CLI --- PRX ==> SRV]")
                v9=(" > " + str(host) + " ")
                v10=("[CLI --- PRX <== SRV]")
                v11=(" > " + str(isOK) + " ")
                v12=(" > " + str(myme) + " " + str(sizu) + "bytes ")
                v13=("[CLI <== PRX --- SRV]")
                v14=(" > " + str(isOK) + " ")
                v15=(" > "  + str(myme) + " " + str(sizu) + "bytes ")
                v16=("[CLI disconnected]")
                v17=("[SRV disconnected]")
```

Here they are being appended manually in order to prevent multithreading from printing them in the wrong order each time.

```
#append all of the strings
                finalLog+=v1
                finalLog+='\n'
                finalLog+=v18
                finalLog+='\n'
                finalLog+=v2
                finalLog+='\n'
                finalLog+=v3
                finalLog+='\n'
                finalLog+=v4
                finalLog+='\n'
```

```
#print them all at once
                print finalLog
```

This method begins the entire process and is thus named so. It manages all arguments and flags, as well as the first socket.

```
#Beginning is the method that deals with all of the system args, it also opens
#the client sockets so that they may send requests to the proxy.
def Begin():
        try:
#inititalize the socket
                print 'initializing'
                global isThreading
                global isProxyCompres
                global mul
                global cp
                if len(sys.argv) > 1:
#if the user has to
                        for i in sys.argv[1:]:
                                if i.isdigit():
                                        serverPort = int(i)
                                if "mt" in i or "MT" in i:
                                        isThreading = True
                                        mul = 'O'
                                if "comp" in i or "COMP" in i:
                                        isProxyCompres = True
                                        cp = 'O'
#make sure they've entered a port
                if len(sys.argv) <= 1:
                        print('please enter a port address arg')
                        sys.exit(2)
#get and save the port for later
#initialize the socket
                myS = socket(AF_INET, SOCK_STREAM)
#allow the same address to be reused
                myS.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
#bind the port
                myS.bind(('', serverPort))
#begin listening for requests
                myS.listen(1)
                print 'server is ready to recieve'
        except Exception, e:
                print e
                sys.exit(2)
#initialize the count variable for connections
        count1 = 1
```

This while loop takes in each new request and decodes their information. It then sends it to the wholeProg function in whichever way specified.

```
#this loop continuously takes in new requests, within this loop it
#also continuously calls the threading function if it is turned on
#otherwise it just calls the driver function as a normal,
#sequential method
#the loops accepts the information from the socket, recieves it and decodes it
#it then sends that info to the wholeProj function/method.
        while True:
                try:
#get Client info in new socket
                        clientConSock,clientAddr = myS.accept()
                        clientInfo = clientConSock.recv(1040).decode()
#if the boolean is set from the opt being requested
                        if isThreading:
#start a new thread with my function correctly sorting the client request data
                                start_new_thread(wholeProg, (clientConSock, clientAd
dr, count1, clientInfo))
#call it like noromal if threading not turned on
                        else: wholeProg(clientConSock, clientAddr, count1, clientInf
o)
                        count1+=1
                except KeyboardInterrupt:
                        myS.close()
                        sys.exit(1)
```

This function sorts all of the client information as described in the comments. It splits everything line by line and does a check within each line to parse the data. It extracts essential elements like the host and port to use later as well and sets the default port to 80.

```
#the function that takes in the clients socket information and address
#This method makes sure the header and body info is correctly sorted, and
#parses through the headers to check or extract desired info like the host
#and port, accept encoding and to remove hop by hop headers
def sortReqData(clientConSock, clientInfo, clientAddr):
        try:
#split the client information
                splitInfo = clientInfo.split('\r\n')
#variable declarations for edited headers and header info
                formattedClientInfo = ""
                host=""
                port=""
                global typu
                global isProxyCompres
#loop through each line in splitInfo
                for line in splitInfo:
                        typu = splitInfo[0]
#split by sections devided by colons and if there are three seperate ones there is a
 non default port
#correctly get the post and host name without spaces
                        if "Host: " in line:
                                item = line.split(":")
#take the host as the first element forward
                                host = item[1]
                                host = host[1:]
                                if len(item) > 2:
                                        port = item[2]
                                else:
#default port 80
                                        port = 80
```

This section is removing all the hop by hop headers as it should and checks if the client supports zipped files. It will only send them those files it the client does. It then goes back to the wholeProg for the next call.

```
#remove the headers that a proxy should
                        if (not "Upgrade"in line) and (not "Keep-Alive: "in line) an
d (not "TE: "in line) and (not "Connection: " in line) and (not "Trailer: "in line)
:
                                formattedClientInfo+=line+'\r\n'

                        if "Accept-Encoding: gzip" not in line:
#if the client doesn't except zipz don't send it
                                proxyCompress = False
#make sure theres two spaces at the end
                formattedClientInfo+='\r\n'
#display it
                print formattedClientInfo
                return host, port, formattedClientInfo

        except KeyboardInterrupt:
                print 'you have chosen to stop the proxy'
                myS.close()
                sys.exit(1)
```

This section is responsible for setting up a second socket and sending the data through it.

```
#This method is also called by the wholeProg, it creates a socket to the server and
sends the clients information and requests
#it also determines weather or not the clients that accept encoding have received ba
ck their zip file and appropriate headers
def proxyToInt(host, port, formattedClientInfo, clientConSock, clientAddr):
#init, connect and send the client info over second socket
        secondS = socket(AF_INET, SOCK_STREAM)
        secondS.connect((host, 80))
        secondS.send(formattedClientInfo)
#variables that store logging data
        global isOK
        global myme
        global sizu
        global isProxyCompres
        bodyReply = None
        isFirst = True
```

This loop receives the data sent back by the server and also sorts it

```
#continueously receive the requested data from the server, then send it back through
 the socket to the client
        while True:
#get the reply from the internet
                intReply = secondS.recv(8192)
#only do header things upon first request
                if isFirst:
                        encodedReply = ""
                        insertContHead = False
                        dontInsert = False
                        headerReply = intReply.split('\r\n\r\n')[0]
#headers split line by line
                        splitBodyReply = headerReply.split('\r\n')
#body seperated from headers
                        bodyReply = intReply.split('\r\n\r\n')[1]
                        for line in splitBodyReply:
#find the line with the Status code
                                if "HTTP/1.1" in line:
                                        isOK = line[9:]
#find the line with the myme type
                                if "Content-Type" in line:
                                        myme = line[14:]
#if the flag is still set bc client accepts zipz
                                if isProxyCompres:
#if this content header is in the response, the file is zipped and can be sent to th
e client that requested it
                                        if "Content-Encoding: gzip" in line:
#if the header is not in the response it is not zipped and should be
                                                insertContHead = False
                                        if "Content-Encoding: gzip" not in line:
                                                insertContHead = True
```

This area checks to see if the server sent back a zip file by looking in the header. If they didn't, we do it for them and fix the header.

```
#if the proxy requests encoding, client is accepting, and the file is not zip
                                if insertContHead:
#add content if its not already there
                                        encodedReply+=line+'\r\n'
#after looping through the header if we should insert the header
                        if insertContHead:
#amake a new header string with the encoding at the end
                                encodedReply+='Content-Encoding: gzip'+'\r\n'
                                encodedReply+='\r\n'
#never do it touch header info for the rest of the body
                        isFirst = False
                zipped =None
                out = None
                FILENAME= None
```

The isFirst Boolean is important as it distuishes the first header set from all the rest of the bodies. The first headers only come once and need to be separated out.
It then actually zips files that need zipping and sends all the appropriate info back through the clients socket as they requested.

```
#if we're done looping through the headers, attach first body from first response to
 the rest of the responses (which only have body data)
                if not isFirst:
                        bodyReply+=intReply
#should zip only the bodies of the rest
                        out = StringIO.StringIO()
                        with gzip.GzipFile(fileobj=out, mode="w") as zipped:
                                zipped.write(bodyReply)
#make sure that the reply is not empty to get len and send
                if (len(intReply) > 0):
                        sizu = len(intReply)
#if the user turned on compress data
                        if isProxyCompres:
                                if insertContHead:
#send correctly adjusted header
                                        clientConSock.send(encodedReply)
#send correctly zipped value
                                        print encodedReply
                                        clientConSock.send(out.getvalue())
                                if not insertContHead:
#send the reply zipped and unseperated as it normally is
                                        clientConSock.send(intReply)
#if the user didn't request compression send it as normal
                        else: clientConSock.send(intReply)
#if the reply no longer has any size close the sockets and end the prog
                else:
                        secondS.close()
                        clientConSock.close()
                        break
#start here
Begin()
```

Working Screenshots with at least 3 distinguishing example cases.

In this case we are calling the program using both multithreading and compression, as well as the specified port.

```
nano@nano-VirtualBox:~/Networks/Proj3/u_2017843374_3$ sudo ./run.sh 390 -mt -com
p
```

Here is the log that shows both flags are successfully on.

```
starting proxy server on port 59504
[O] MT | [O] COMP | [X] CHUNK | [X] PC
-----------------------------------------
1
[CLI connected to 127.0.0.1:59504]
[CLI ==> PRX --- SRV]
 > GET http://yscec.yonsei.ac.kr/course/view.php?id=111580 HTTP/1.1
[CLI connected to yscec.yonsei.ac.kr:80]
[CLI --- PRX ==> SRV]
 > yscec.yonsei.ac.kr
[CLI --- PRX <== SRV]
 > 303 See Other
 > text/html 884bytes
[CLI <== PRX --- SRV]
 > 303 See Other
 > text/html 884bytes
[CLI disconnected]
[SRV disconnected]

GET http://yscec.yonsei.ac.kr/login/index.php HTTP/1.1
Host: yscec.yonsei.ac.kr
```

Here is an example of the log with only compression on working succesfully

```
starting proxy server on port 46562
[X] MT | [O] COMP | [X] CHUNK | [X] PC
-----------------------------------------
1
[CLI connected to 127.0.0.1:46562]
[CLI ==> PRX --- SRV]
 > GET http://cs.yonsei.ac.kr/ HTTP/1.1
[CLI connected to cs.yonsei.ac.kr:80]
[CLI --- PRX ==> SRV]
 > cs.yonsei.ac.kr
[CLI --- PRX <== SRV]
 > 200 OK
 > text/html; charset=utf-8 728bytes
[CLI <== PRX --- SRV]
 > 200 OK
 > text/html; charset=utf-8 728bytes
[CLI disconnected]
[SRV disconnected]

GET http://cs.yonsei.ac.kr/css/style.css HTTP/1.1
Host: cs.yonsei.ac.kr
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0
```

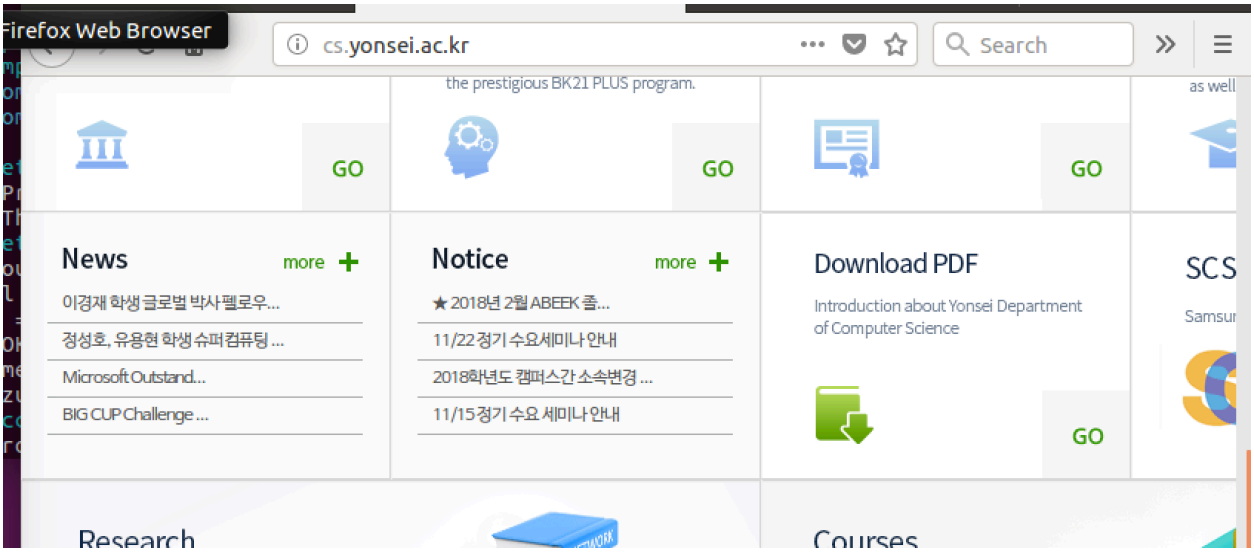
Here is the log with only multithreading on working successfully.

```
Accept-Encoding: gzip, deflate
Referer: http://cs.yonsei.ac.kr/
Cookie: _ga=GA1.3.551996551.1509023042; _gid=GA1.3.1852153615.1511363467; PHPSES
SID=7vbgrttg5ts1dm9dn5iiuem0i0
If-Modified-Since: Sun, 31 Jan 2016 02:10:09 GMT
If-None-Match: "19cc9b2-16b0-56ad6d01"



starting proxy server on port 60980
[O] MT | [X] COMP | [X] CHUNK | [X] PC
----------------------------------------
20
[CLI connected to 127.0.0.1:60980]
[CLI ==> PRX --- SRV]
 > GET http://cs.yonsei.ac.kr/img_en/box_banner02.gif HTTP/1.1
[CLI connected to cs.yonsei.ac.kr:80]
[CLI --- PRX ==> SRV]
 > cs.yonsei.ac.kr
[CLI --- PRX <== SRV]
 > 304 Not Modified
 > text/html; charset=UTF-8 135bytes
[CLI <== PRX --- SRV]
 > 304 Not Modified
 > text/html; charset=UTF-8 135bytes
```

This call demonstrates that the port can be changed in any order as well as the case insensitive flags.

```
nano@nano-VirtualBox:~/Networks/Proj3/u_2017843374_3$ sudo ./run.sh -mt 394
initializing
server is ready to recieve
GET http://cs.yonsei.ac.kr/ HTTP/1.1
Host: cs.yonsei.ac.kr
```

Here is the Yonsei website successfully loaded from all three cases



References:

Stack overflow many many times. Thread and socket import librarys. Network Programming textbook, Peirsons. Github. Mnots blog: what proxies must do.
https://stackoverflow.com/questions/1265665/python-check-if-a-string-represents-an-int-without-using-try-except
https://stackoverflow.com/questions/43810467/python-ftplib-socket-gaierror-errno-3-temporary-failure-in-name-resolution
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding