Ryerson University

Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Networks
Course Number:	COE768
Semester/Year (e.g.F2016)	S2020

Instructor:	Baha Uddin Kazi

Assignment/Lab Number:	Project
Assignment/Lab Title:	P2P Application

Submission Date:	09/08/2020
Due Date:	09/08/2020

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Zou	Henry	500628304	1	
Ghiasi	Ahmad	500780809	1	

Reset Form

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Table of Contents

Introduction

The project is about implementing a P2P application. As we know, a network application can be a both a client and a server. So, whenever Peer A wants to make one of its files available for download, what will do is that it will register content to server. Once, content is registered, Peer A will become server of content. Another peer which in this case is now Peer B wants to download content will first call server and get address of Peer A. Afterwards, Peer B acts as a client, then downloads content from Peer A. Additionally, after downloading, Peer B will register content to server which becomes the server as well.

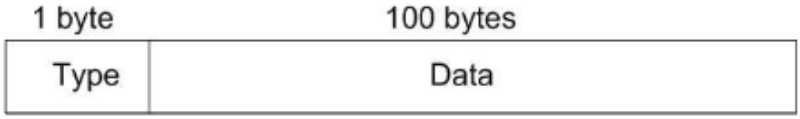
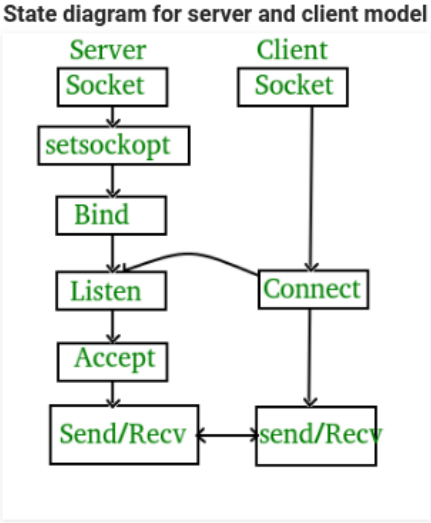


Figure 1. PDU exchanged among peers and index server has following format above

As we know, the type field indicates the PDU type. The data field has the data. As it was mentioned in the lab manual, there are eight PDU types.

Background information on socket programming would be, it is a way of connecting two nodes on a network to communicate with each other. What happens next is that, one socket or a node listens on a particular port at an IP and the other node reaches out to the other to form a connection. Server makes or forms the listener socket while client reaches to the server.



Stages for Server:

- Socket creation: int sockfd = socket(domain, type, protocol)
- Domain: integer, communication domain. (Ex, AF_INET [IPv4 protocol])
- Type: communication type SOCK_STREAM: TCP(reliable, connection oriented), SOCK_DGRAM:UDP(unreliable, connectionless)
- Protocol: value for Internet Protocol(IP) which is 0. This is the same number which shows on protocol field in the IP header of a packet.
- Bind: s.bind((host, port))
- Listen: s.listen(5)
- Accept: conn, addr = s.accept()

Stages for Client:
Socket connection: It is exactly same as server's socket creation
Connect: s = socket.socket(socket.SOCK_DGRAM)

Description of Client and Server programs

There are nine PDU types. Every time a new client would connect to the server, the server would fork and make a new child process to service the connection. The client has a main process loop which services the user, send the PDU and would fork to listen for downloads.

1. Content Registration

Type 'R': This type, what it does is that, a peer can register its content to server by sending an R-type PDU using UDP. Data portion of PDU obtains peer name, content name and address which would be the IP address plus port number where content can be downloaded. The image below shows the format.

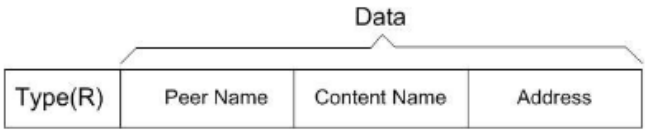


Figure 2. Representation of type 'R'

When index server receives an R-type PDU, what it will do first is that, it will check if another peer with same name registered same content name. If in this case, if it happens, server will send an E-type PDU to tell peer to choose another peer name. So, now if there

won't be any issue of peer name, content server will register content and store. Afterwards, it will send an A-type PDU to give it acknowledge the registration. Before all this, make sure that peer makes a TCP socket for content download.

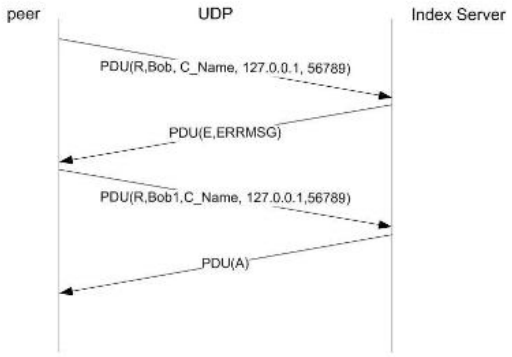


Figure 3. Registration procedure performed between a peer and index server

2. Content Download

Type 'S': This type, what it does is that, a peer at first contacts server to search for address of content server. It will perform this operation by using the format below. Server responds back with either a S-type or E-type, S-type for containing address of a content server and E-type which says that no such content is available. If a S-type PDU is received, it will take address from PDU and setup a TCP connection with server. If TCP connection is established, peer sends a D-type PDU to server to download. If content is ready, server will deliver content by sending a consecutive C-type PDUs which has content. Once it is downloaded, content will be registered to server, hence becomes the serve of content. In some cases, there is a possibility that server could have more than one server for a given content. So, now in this case, in order to send out equally, server will always have to pick recent registered server for download request.

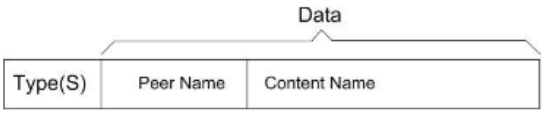


Figure 4. S-type PDU

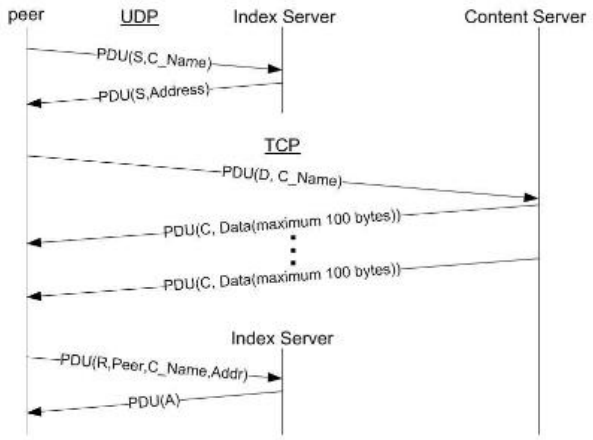


Figure 5. PDU transaction for Content Downloading and the Subsequent Content Registration.

3. Content Listing

Type 'O': What does this type does, it lists the contents registered. So, by sending an O-type PDU to index server, it will respond with an O-type PDU that have the list of registered content.

4. Content De-Registration

Type 'T': What does this type does is that, it provides the client an option to allow a peer to de-register content. In order to do that, peer will send a T-type PDU to server to do the operation.

Observations and Analysis

For observation, when I used to select first for the client side, but it would make the client code slow, so I used a fork to speed up the execution side. The server side could not run without a fork because it must be connected to two clients at the same time and it must listen for new connections and service 2 clients. The problem with this program is because the program is forking the fList between two child processes have different memory spaces so when client registers files, it does not appear on the other process list since they have different child process processing it. An extra final pdu was needed to be added instead of the eight because the client needs to know when the file ends or it will be forever waiting on the next data, if the uploading client does not tell it to stop.

The images shown below indicate the all the situation with each PDU ;

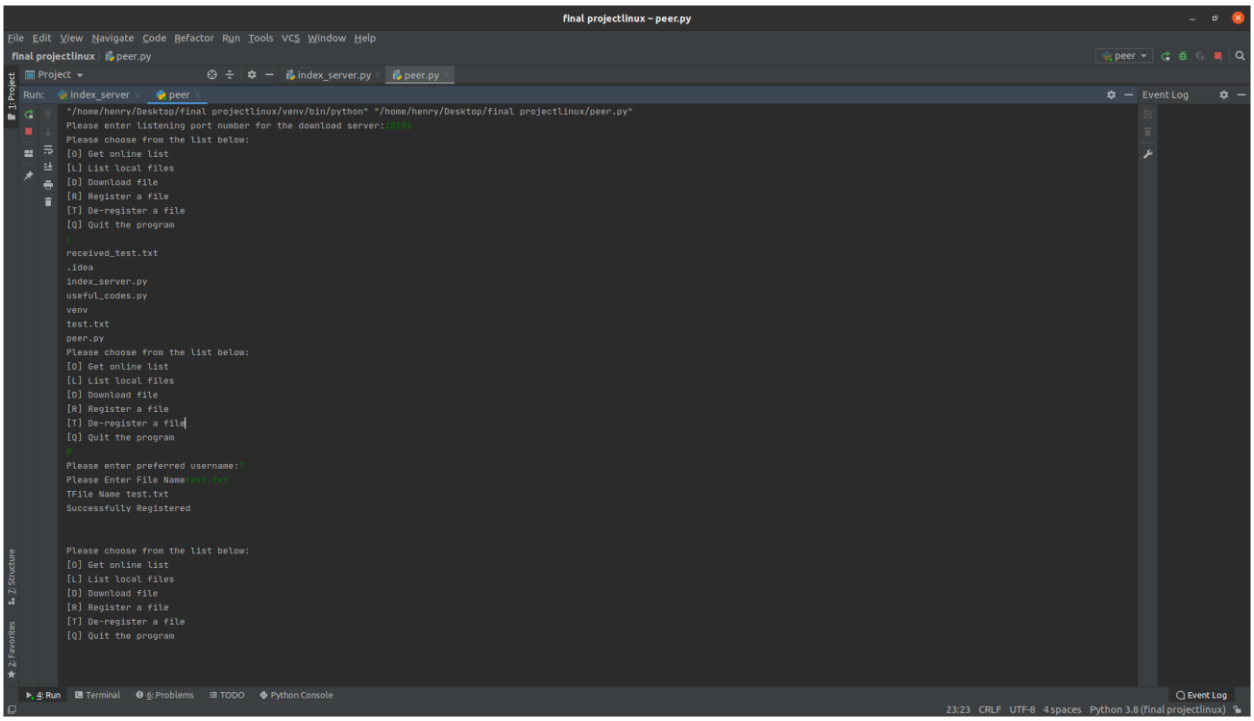


Figure 6. Indicating the 'L' & 'R' type (Client Side)

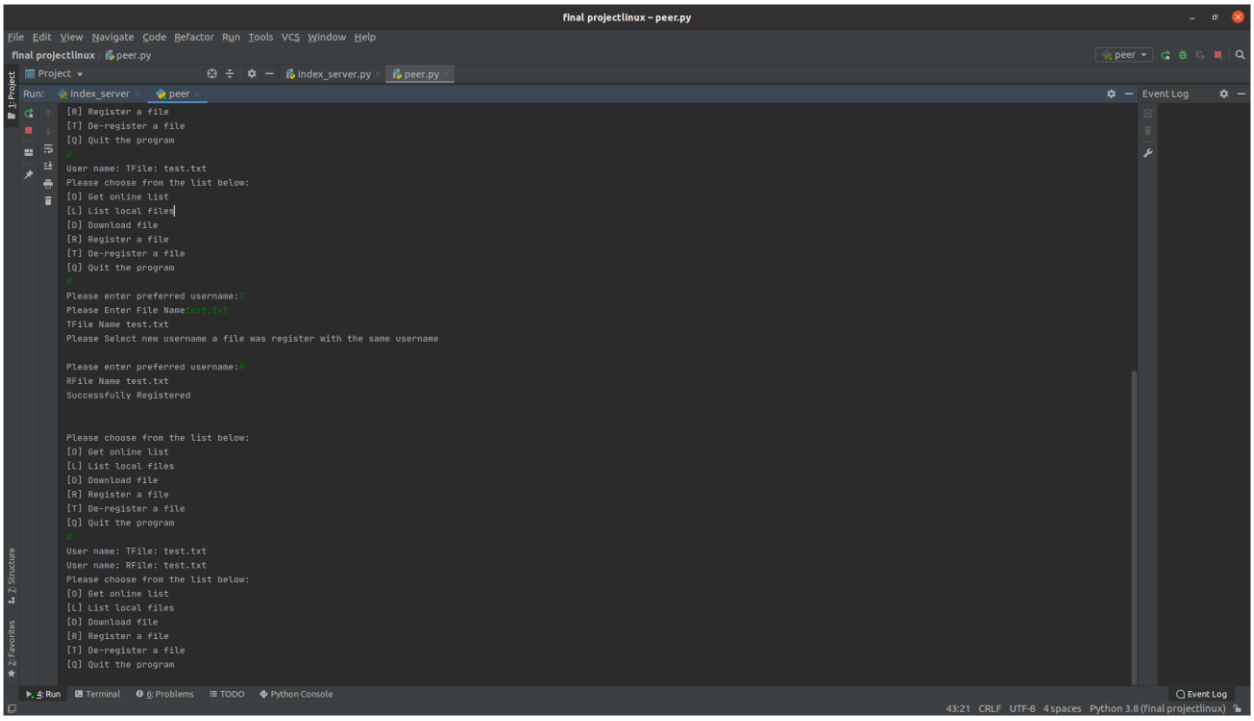


Figure 7. Demonstrating 'O' PDU type (Client Side)

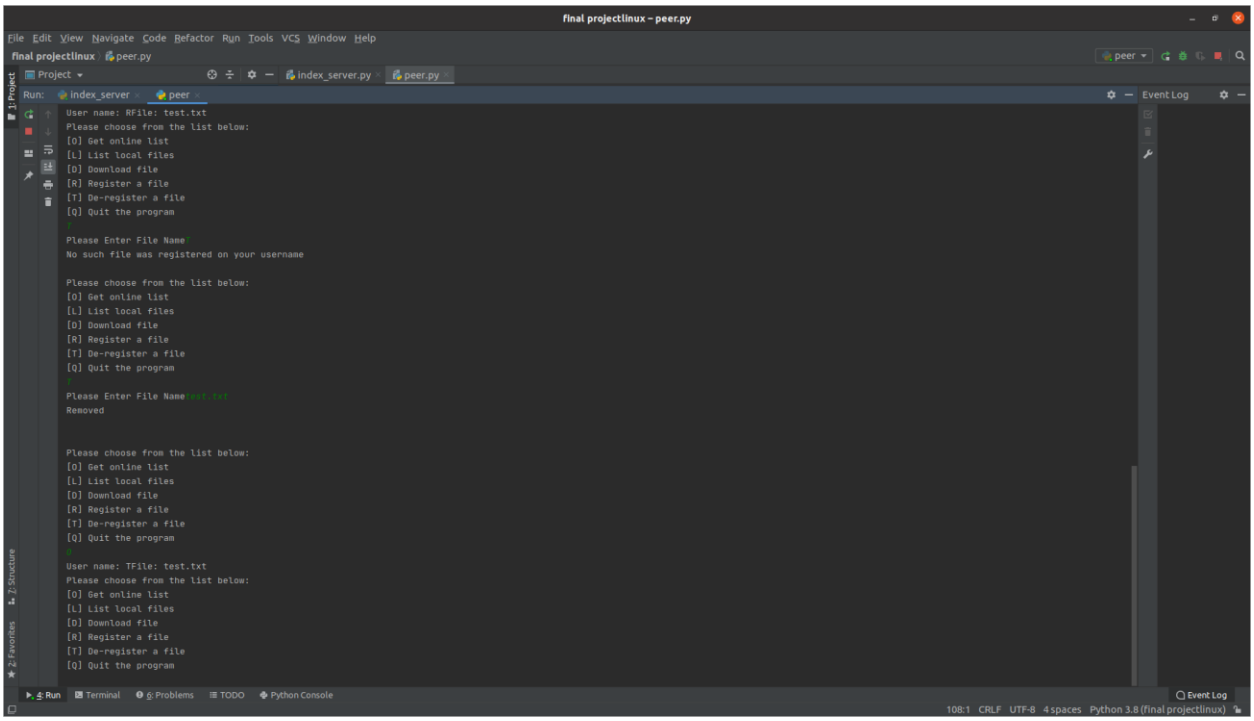


Figure 8. Demonstration of 'T' type (Client Side)

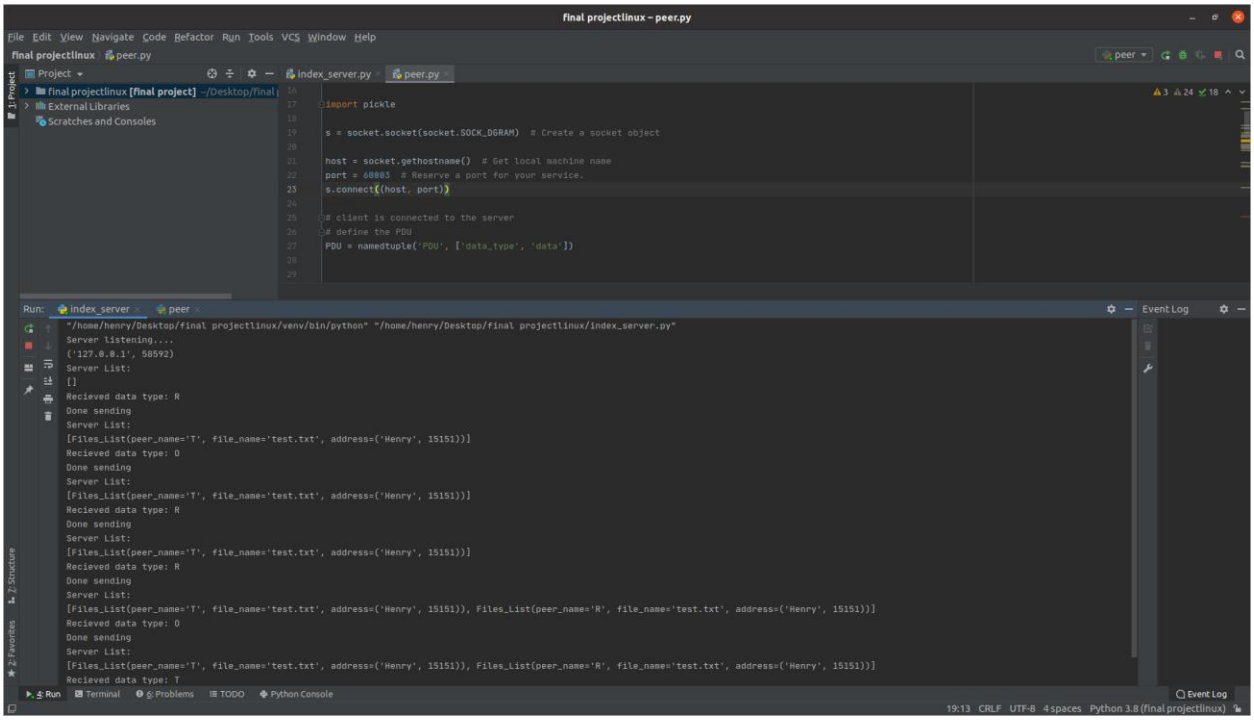


Figure 9. Server Responding Back (Server Side)

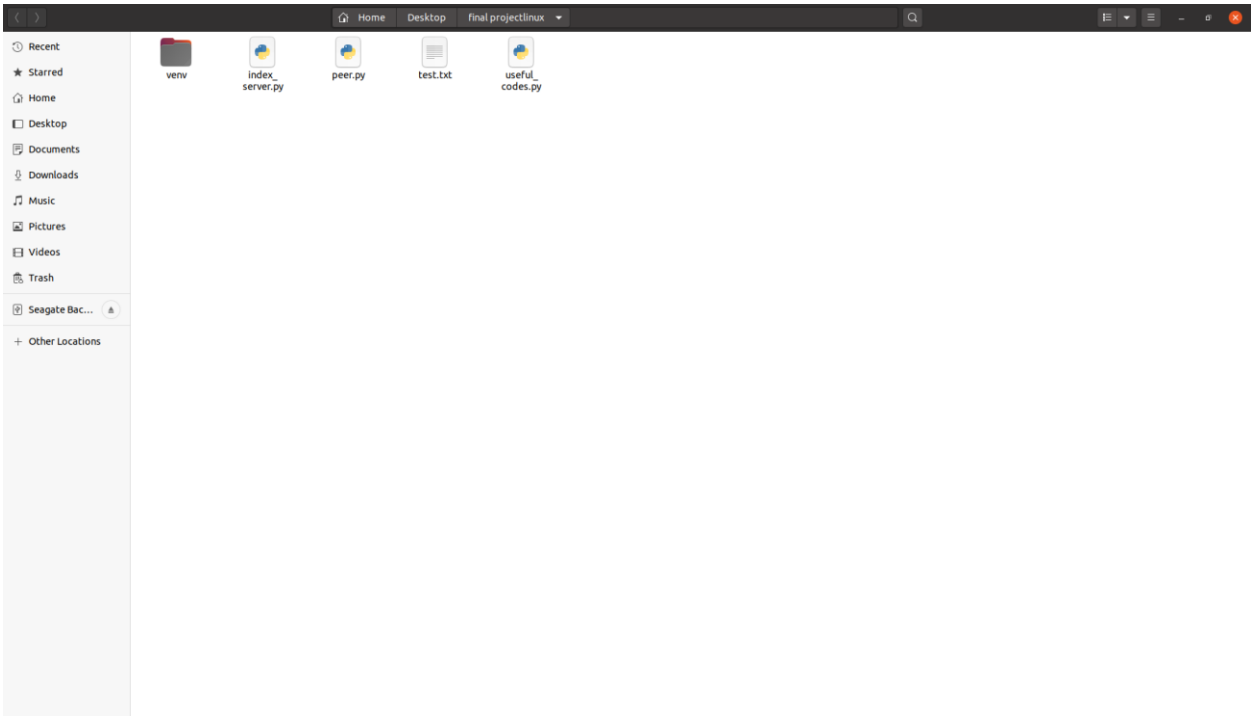


Figure 10. Before 'Receiving test file'

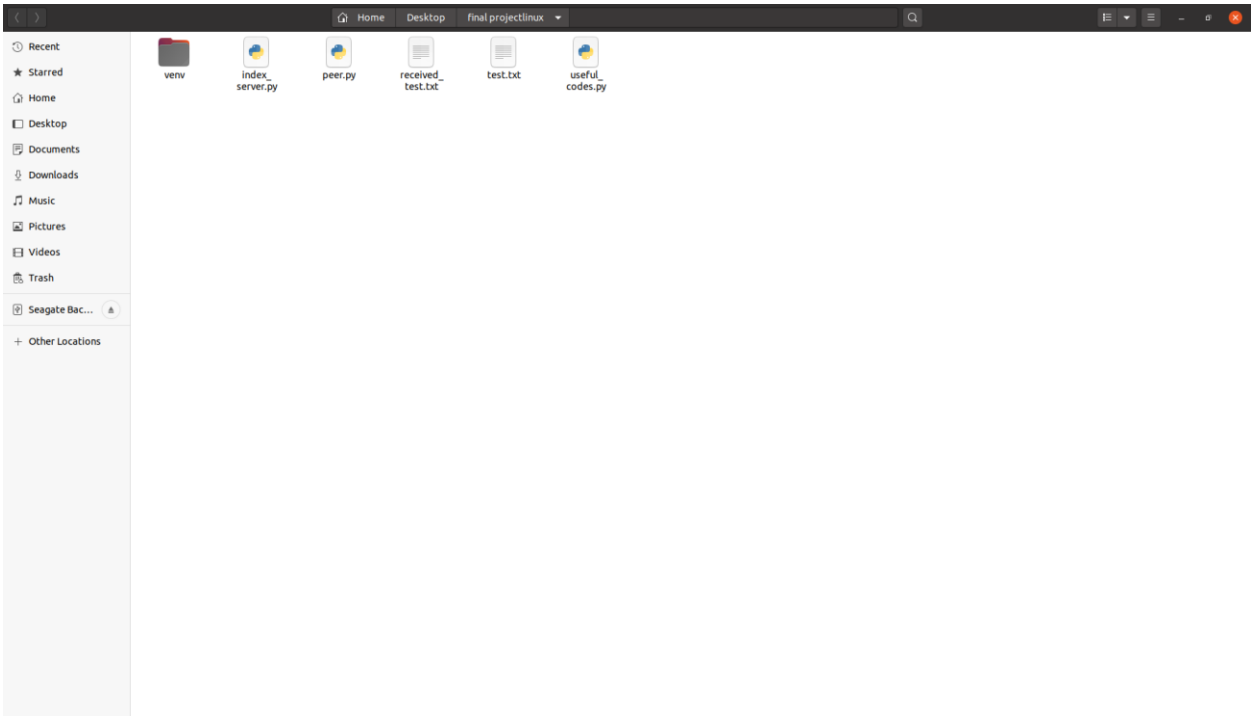


Figure 11. After 'Receiving test file'

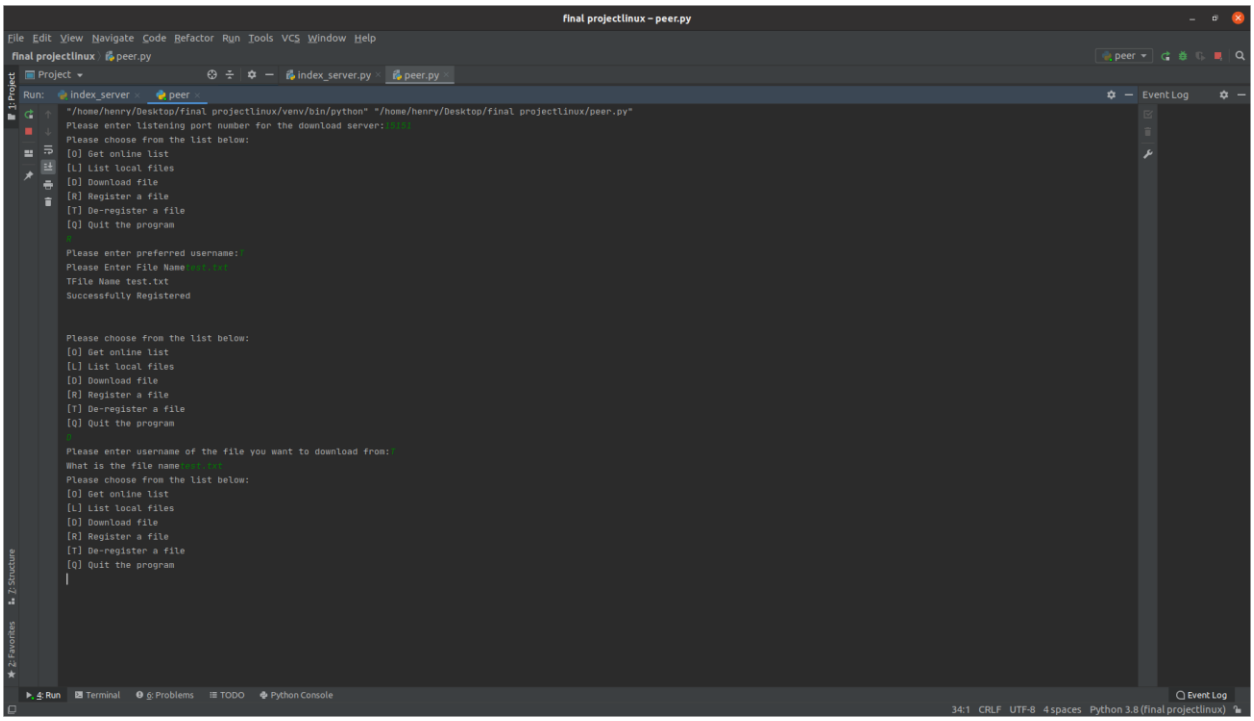


Figure 12. Performing the PDU Types in Order to Achieve the Images Shown Above

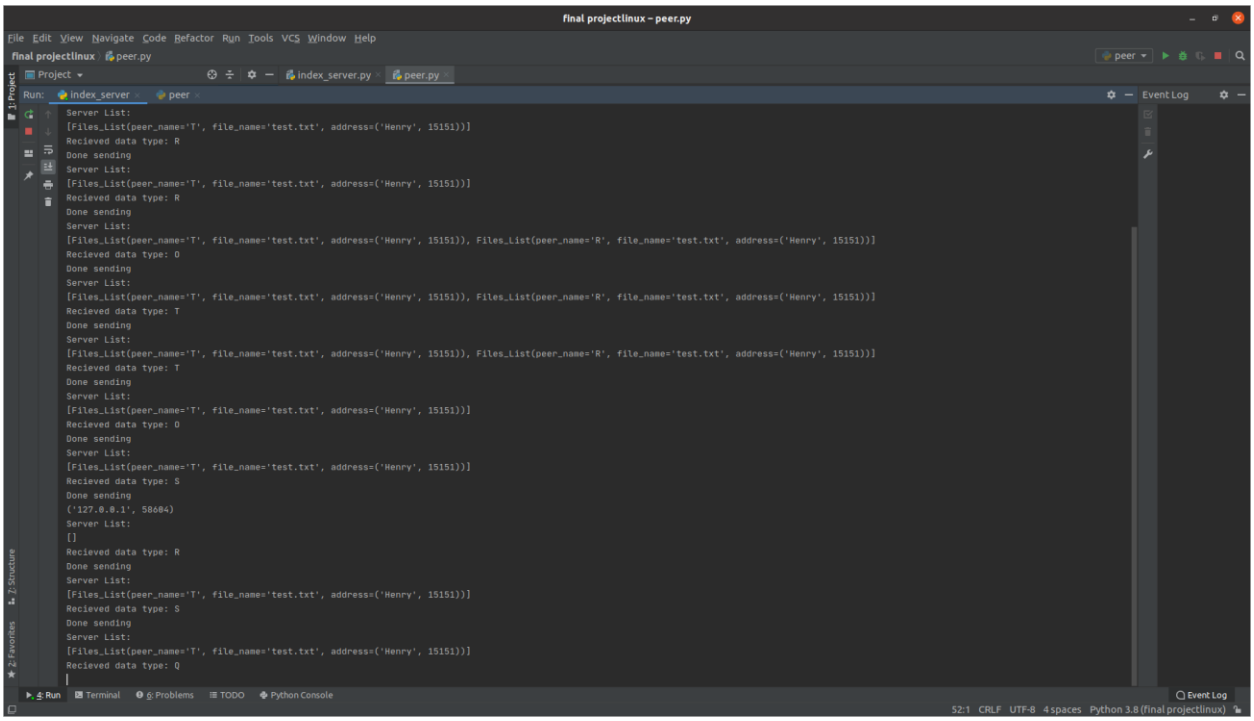


Figure 13. Server Side Responding back

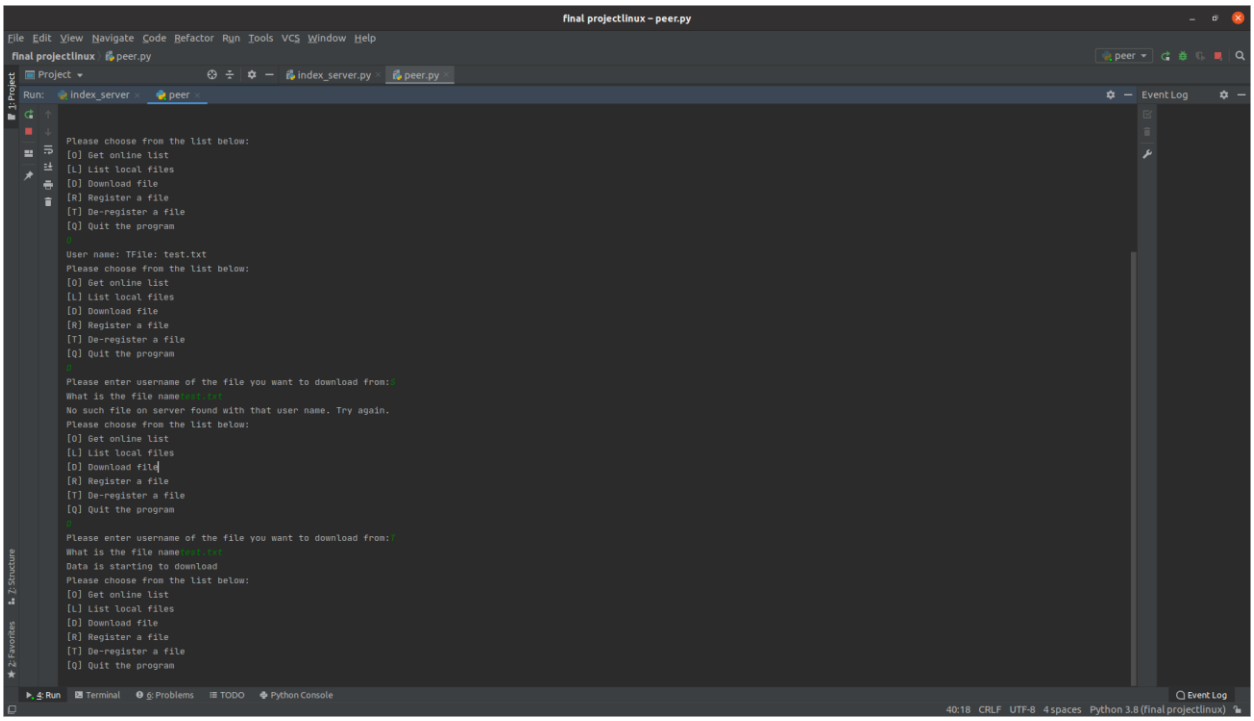


Figure 13. Lastly, Performing the 'O' & 'D' PDU Type

Conclusions

In conclusion, a P2P application was successfully implemented. The server is capable of handling multiple file transfers and the client can manage both uploading and downloading files from the server. As per the requirement the PDU formats mentioned in the lab manual were used and designated tasks associated with each PDU were successfully performed. In short, a working P2P application was implemented and the basic concept of P2P mechanism was clearly understood.

References

[1] COE 768 Project 1 Manual, "*P2P_Project*". Ryerson University. August 8, 2020. [Online]. Available: Ryerson D2L portal.

Appendix

Index_Server.py

```
# index_server.py
'''
Index Server
Message types:
R - used for registration
A - used by the server to acknowledge the success
Q - used by chat users for de-registration
D - download content between peers (not used here)
C - Content (not used here)
S - Search content
E - Error messages from the Server
'''

import socket  # Import socket module
from collections import namedtuple
import pickle
import os

port = 60003 # Reserve a port for your service.
s = socket.socket(socket.SOCK_DGRAM)  # Create a socket object
host = socket.gethostname()  # Get local machine name
s.bind((host, port))  # Bind to the port
s.listen(5)  # Now wait for client connection.
# server is up and listening
print('Server listening....')

PDU = namedtuple('PDU', ['data_type', 'data'])
Files_List = namedtuple('Files_List', ['peer_name', 'file_name', 'address'])
fList = []  # list of files, containing Files_List namedtuples


def service():
    while True:
        binary_pdu = conn.recv(1024)
        print("Server List:")
        print(fList)
        # receiving the binary_pdu = conn.recv(100)
        # convert pdu from binary to pdu object using pickle
        pdu = pickle.loads(binary_pdu)
        # extract the type from pdu, type = pdu.data_type
        # check data_type0
        data_type = pdu.data_type
        data = pdu.data
        print("Recieved data type: "+data_type)
        # if 'R'
        # check list of files
        # if file does not already exist (new)
        # create new Files_List object
        # send 'A' type pdu
        # send 'E' type pdu
        if data_type == 'R':
            p_peer_name = data.get('peer_name')
            p_file_name = data.get('file_name')
            p_peer_address = data.get('address')
            already_exists = False
            for i in fList:
                if i.peer_name == p_peer_name and i.file_name == p_file_name:
                    e_pdu = PDU('E', {'msg': 'File already exists'})
```

```
                b_pdu = pickle.dumps(e_pdu)
                conn.send(b_pdu)
                already_exists = True
                break
        if not already_exists:
            file = Files_List(p_peer_name, p_file_name, p_peer_address)
            fList.append(file)
            a_pdu = PDU('A', {'msg': 'Successfully Registered'})
            b_pdu = pickle.dumps(a_pdu)
            conn.send(b_pdu)
    # else if 'S'
    # check the fList
    # if the file exists, send the file
    # else send 'E' pdu
    elif data_type == 'S':
        p_peer_name = data.get('peer_name')
        p_file_name = data.get('file_name')
        found_content = False
        for d in fList:
            if d.peer_name == p_peer_name and d.file_name == p_file_name:
                target = d
                found_content = True
                break
        if found_content:
            pdu = PDU('A', target.address)
            b_pdu = pickle.dumps(pdu)
            conn.send(b_pdu)
        else:
            e_pdu = PDU('E', {'msg': 'Record does not exists on the list.'})
            b_pdu = pickle.dumps(e_pdu)
            conn.send(b_pdu)
    elif data_type == 'O':
        menu = []
        for i in fList:
            menu.append((i.peer_name, i.file_name))
        pdu = PDU('O', menu)
        b_pdu = pickle.dumps(pdu)
        conn.send(b_pdu)
    elif data_type == 'T':
        p_peer_name = data.get('peer_name')
        p_file_name = data.get('file_name')
        found_content = False
        for d in fList:
            if d.peer_name == p_peer_name and d.file_name == p_file_name:
                target = d
                found_content = True
                break
        if found_content:
            fList.remove(target)
            pdu = PDU('A', target.address)
            b_pdu = pickle.dumps(pdu)
            conn.send(b_pdu)
        else:
            e_pdu = PDU('E', {'msg': 'Record does not exists on the list.'})
            b_pdu = pickle.dumps(e_pdu)
            conn.send(b_pdu)
    elif data_type == 'Q':
        p_peer_name = data.get('peer_name')

        deletelist = []
        for x in fList:

            if x.peer_name == p_peer_name:
```

```
                    deletelist.append(x)

            for i in deletelist:
                fList.remove(i)


            pdu = PDU('A', "Quit")
            b_pdu = pickle.dumps(pdu)
            conn.send(b_pdu)
            s.close()
            conn.close()
            exit(0)
        # else if 'T'
        # remove the file from list
        # if file does not exist send 'E'
        # if removed successfully send 'A'

        # else if 'O'
        # iterate through the fList
        # create a list of files
        # sned the 'O' pdu, data is the list

        # else ....
        print('Done sending')


while True:
    conn, addr = s.accept()  # Establish connection with client.
    newpid = os.fork()
    if newpid == 0:
        service()
    else:
        print(addr)
# conn.close() # close the connection
```

Peer.py

peer.py

```
# peer.py
'''
A P2P client
It provides the following functions:
- Register the content file to the index server (R)
- Contact the index server to search for a content file (D)
    - Contact the peer to download the file
    - Register the content file to the index server
- De-register a content file (T)
- List the local registered content files (L)
- List the on-line registered content files (O)
'''
import os
import socket  # Import socket module
from collections import namedtuple

import pickle

s = socket.socket(socket.SOCK_DGRAM)  # Create a socket object

host = socket.gethostname()  # Get local machine name
port = 60003  # Reserve a port for your service.
s.connect((host, port))

# client is connected to the server
# define the PDU
```

```
PDU = namedtuple('PDU', ['data_type', 'data'])


# Functions
def select_name():
    return input('Please enter preferred username:')


def download_file(filename, address):
    rec_port = socket.socket()  # this is a TCP connection
    rec_port.connect(address)
    d_pdu = PDU('D', {'file_name': filename})
    b_pdu = pickle.dumps(d_pdu)
    rec_port.send(b_pdu)
    rec_filename = 'received_{}'.format(filename)
    with open(rec_filename, 'wb') as f:  # receive open file with received_file
        while True:
            message = rec_port.recv(1024)  # get pdu reply
            pdu = pickle.loads(message)  # decode binary pdu
            typepdu = pdu.data_type  # get PDU data type

            data = pdu.data  # get PDU data
            if (typepdu == 'D' or typepdu == 'F'):
                f.write(data.encode())  # write data to file convert to binary
            elif (typepdu == 'E'):
                print('Error detected')  # Error detected
                break
            if (typepdu == 'F'):  # After final pdu
                rec_port.close();
                f.close();
                break;
    return


def download(socket):
    pdu = socket.recv(1024)  # receive the request
    fpdu = pickle.loads(pdu)
    data = fpdu.data
    filename = data.get('file_name')
    with open(filename, 'r') as reader:  # open file with file name
        line, nextline = reader.read(100), reader.read(100)  # read line with a buffer
to check if it is the file name max pdu size is 60 because of pickle overhead
        while True:  # infinte loop
            if (nextline):  # if next line

                namepdu = PDU('D', line)  # create a data pdu
                binarypdu = pickle.dumps(namepdu)  # create binary pdu
                socket.send(binarypdu)  # send binary pdu
                line = nextline  # change next to this line
                nextline = reader.read(100)  # read next line
            else:  # if its the final

                namepdu = PDU('F', line)  # create a data pdu
                binarypdu = pickle.dumps(namepdu)  # create binary pdu
                socket.send(binarypdu)  # send binary pdu
                reader.close()  # close the file
                break



    return
```

```
def listen():
    while True:
        fileReq_Socket, fileReq_addr = ss.accept()  # accept connection
        newpid = os.fork()
        if newpid == 0:
            download(fileReq_Socket)
        else:
            print('Peer connection accepted')
        # check the file name (it should be 'D' type)
        # send the file using 'C' type
        # if file doest not exist send 'E' pdu
        # there is no incomin`  g connection request, so go to the menu and ask the
user for command


# create a server to listen to the file requests
'''
Here we config the server capability of the peers. As a server we need to specify ip
address and ports. Since all the
peers are inside the local network (IP=127.0.0.1), we need to use unique port numbers
for each peers so they can
bind socket successfully. This can be done by generating random numbers and using
try/except command to bind a socket.
Withing multiple attempt we can be sure that peer would eventually bind a socket with
random port number. Here I do not
use this approach. Instead I asked the user to enter a port number manually. During
the test, for each of the peers,
you will need to enter different port numbers for different peers.
The '' for IP address means our server is listening to all IPs,
you can change it to socket.hostname instead like before.'''
inputs = []
outputs = []
exp = []
ss = socket.socket()  # this is a TCP connection
serverPort = int(input('Please enter listening port number for the download server:'))
try:
    ss.bind(('', serverPort))
except Exception:
    pass

ss.listen(5)
inputs.append(ss)
filename = 'Henry'
username = 'Henry'
exp.append(ss)

# service loop

'''
readable, writable, exceptional = select.select(inputs, outputs, exp, 1)
for sock in readable:  # check the incoming connection requests
    if sock is ss:
        fileReq_Socket, fileReq_addr = ss.accept()  # accept connection
        pdu = ss.recv(1024)  # receive the request
        print(pdu)
        # check the file name (it should be 'D' type)
        # send the file using 'C' type
        # if file doest not exist send 'E' pdu
        # there is no incomin`  g connection request, so go to the menu and ask the
user for command
'''
newpid = os.fork()
if newpid == 0:
```

```
    listen()
else:
    while True:
        command = str(input('Please choose from the list below:\n'
                            '[O] Get online list\n'
                            '[L] List local files\n'
                            '[D] Download file\n'
                            '[R] Register a file\n'
                            '[T] De-register a file\n'
                            '[Q] Quit the program\n'))
        if command == 'O':
            o_pdu = PDU('O', {'Content List'})
            b_pdu = pickle.dumps(o_pdu)
            s.send(b_pdu)
            o_pdu = s.recv(1024)
            pdu = pickle.loads(o_pdu)
            data = pdu.data
            for i in data:
                name, file = i
                print('User name: ' + name + 'File: ' + file)

        if command == 'L':
            # list local files
            local_file = os.listdir('.')

            for name in local_file:
                print(name)
        if command == 'D':
            user = input('Please enter username of the file you want to download
from:')
            file_name = input('What is the file name')
            s_pdu = PDU('S', {'peer_name': user, 'file_name': file_name})
            b_pdu = pickle.dumps(s_pdu)
            s.send(b_pdu)
            o_pdu = s.recv(1024)
            pdu = pickle.loads(o_pdu)
            address = pdu.data
            if pdu.data_type == 'A':
                print("Data is starting to download")
                download_file(file_name, address)

            elif pdu.data_type == 'E':
                print("No such file on server found with that user name. Try again.")
                # extract data_type
            # send 'O' type pdu
            # receive the list
            # print the list
            # ask user for the target file
            # create 'S' type pdu
            # send 'S' pdu to the index server
            # receive 'S' pdu in response
            # extract address
            # establish new connection to the peer

        if command == 'R':
            # get the file name
            # create 'R' pdu using username, filename, IPaddress and portnumber
            # send 'R' pdu
            # receive response pdu
            # if 'A', done
            # else if 'E',
            username = select_name()
            filename = str(input('Please Enter File Name'))
```

```
            while True:
                if os.path.isfile(filename):
                    break
                filename = str(input('Please Enter Valid local File Name'))
            while True:
                print(username + "File Name " + filename)
                r_pdu = PDU('R', {'peer_name': username, 'file_name': filename,
'address': (host, serverPort)})
                b_pdu = pickle.dumps(r_pdu)
                s.send(b_pdu)
                binary_pdu = s.recv(1024)
                pdu = pickle.loads(binary_pdu)
                data = pdu.data
                if pdu.data_type == 'A':
                    print(data.get('msg'))
                    print('\n')
                    break
                elif pdu.data_type == 'E':
                    print('Please Select new username a file was register with the
same username\n')
                    username = select_name()
                    # extract data_type
        if command == 'T':

            filename = str(input('Please Enter File Name'))
            t_pdu = PDU('T', {'peer_name': username, 'file_name': filename, 'address':
(host, serverPort)})
            b_pdu = pickle.dumps(t_pdu)
            s.send(b_pdu)
            binary_pdu = s.recv(1024)
            pdu = pickle.loads(binary_pdu)
            data = pdu.data
            # get the file name from user
            if pdu.data_type == 'A':
                print("Removed")
                print('\n')
            elif pdu.data_type == 'E':
                print('No such file was registered on your username\n')
        if command == 'Q':
            # for all the registered files:
            r_pdu = PDU('Q', {'peer_name': username, 'file_name': filename, 'address':
(host, serverPort)})
            b_pdu = pickle.dumps(r_pdu)
            s.send(b_pdu)
            binary_pdu = s.recv(1024)
            pdu = pickle.loads(binary_pdu)
            data = pdu.data
            exit(0)

# quit the program
```