MAKERERE UNIVERSITY

COCIS

BSE 3208: Distributed Systems Development

Exercise 2 (PEER TO PEER)

CRANE CLOUD

GROUP E-K

GROUP MEMBERS

NO.	**NAME**	**REG NO.**
1	MUGIRE BRUNO	18/U/23422/EVE
2	BENGANA ANTHONY	17/U/3569/EVE
3	SSEBALAMU RONALD GGAANYA	18/U/23401/EVE

GITHUB LINK: https://github.com/theebruno/cranecloud

INTRODUCTION:

Peer-to-Peer(P2P) Technologies are being widely used for sharing the data between the servers and the clients. One of the major technology for file sharing that is implemented nowadays is the Napster-Style Peer-to-Peer File Sharing System.

The older versions of the systems used to have a single server which stores the files in its directory that are received from the clients. The major drawback of these systems was that if a new file has been created in one of the peers, it must be transferred to the server before another peer can access it, which delays the process of transfer from one peer to another. This can be conquered using the Napster system which allows the peer to peer file transfer.

SYSTEM REQUIREMENTS:

- Python 3.7.0(3.5+)
- Command line terminal

DESIGN:

The entire project is designed using Python where we have used the concepts of Socket Programming and Multi-threading. For establishing the connections between the Server and the Clients, I have used TCP/IP protocol using the sockets.

Major Components of the Project:

- Server
- Client

Server (Central Index Server):

This server indexes the content of all the peers (i.e., Clients) that register with it. It also provides add, list, search, download facilities to peers.

Client:

As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file using the necessary command.

Major function of the peer:

- Download

As a server, the peer waits for requests from other peers and sends the requested file when receiving a request. The Peers (i.e., Clients) here, act as both the client and the server. This central index server indexes the files. The peer acts a client to download the files from other peers into its directory.

The peers provide the following interface to the users:

1. Add – registers the file into the server.
2. Lookup– searches the server for a file and returns the list of Clients.
3. Download – downloads the file from another Client.
4. List All – lists all the files from another Clients.
5. Shutdown – disconnects the Client from the network.

POSSIBLE IMPROVEMENTS:

Could develop a User Interface

EXECUTION OF THE PROJECT:

Connecting

Server:

We Start our execution by executing the Server –

```
C:\dollar\master>server.py
Server P2P-CI/1.0 is listening on port 7734
127.0.0.1:63377 connected
```

Now Server will open its Socket and waits for the Clients to get

connected

Client:

We will register 3 clusters(Peers) on to the Server as per the project requirement

Each cluster has his/her own storage space. Since we are testing multiple clusters on a single machine, the `client.py` file should be copied to several different directories to simulate real situation i.e. mkango, mpologoma and chui

For mpologoma cluster –

```
C:\dollar\master\mpologoma>client.py
Connecting to the server localhost:7734
Connected
Listening on the upload port 50806

1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request:
```

For mkango cluster -

```
C:\dollar\master\mkango>client.py
Connecting to the server localhost:7734
Connected
Listening on the upload port 50808

1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request:
```

For chui cluster–

```
C:\dollar\master\chui>client.py
Connecting to the server localhost:7734
Connected
Listening on the upload port 50810

1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request:
```

For server

```
Server P2P-CI/1.0 is listening on port 7734
127.0.0.1:50805 connected
127.0.0.1:50807 connected
127.0.0.1:50809 connected
```

Adding files to be shared in this test-

Mpologoma cluster

```
1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request: 1
Enter the RFC number: 1
Enter the RFC title: rfc1
rfc\rfc1.txt
Recieve response:
P2P-CI/1.0 200 OK
RFC 1 rfc1 BRUNO 50806
```

Listing available files –

Chui cluster

```
1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request: 3
Recieve response:
P2P-CI/1.0 200 OK
RFC 1 rfc1 BRUNO 50806
RFC 2 rfc2 BRUNO 50808
```

Searching available files which shows who has the file.

Mkango cluster

```
1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request: 2
Enter the RFC number: 1
Enter the RFC title(optional): rfc1
Recieve response:
P2P-CI/1.0 200 OK
RFC 1 rfc1 BRUNO 50806
```

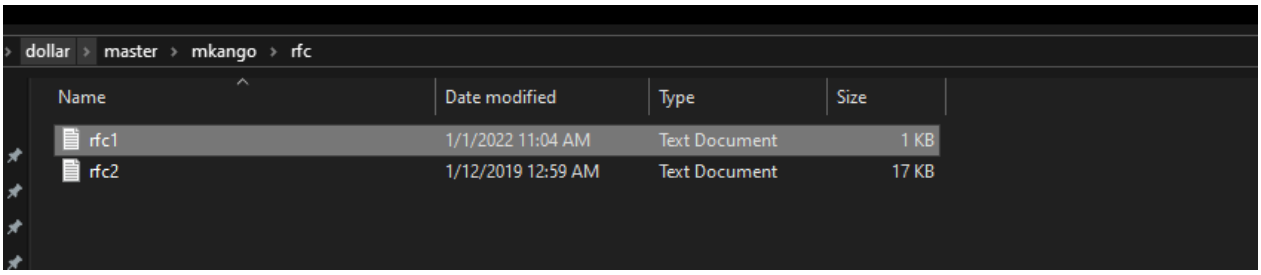
Downloading a file.

In this case from mpologoma cluster to mkango cluster

```
1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request: 4
Enter the RFC number: 1
Available peers:
1: BRUNO:50848
Choose one peer to download: 1
Recieve response header:
P2P-CI/1.0 200 OK
Data: Sat, 01 Jan 2022 08:04:49 GMT
OS: Windows-10-10.0.18362-SP0
Last-Modified: Fri, 11 Jan 2019 21:59:25 GMT
Content-Length: 13
Content-Type: text/plain

Downloading...
Downloading Completed.
Sending ADD request to share...
rfc\rfc1.txt
Recieve response:
P2P-CI/1.0 200 OK
RFC 1 rfc1 BRUNO 50850
```

Here the rf1.txt from mpologoma is downloaded to the mkango server



Leaving network

Mkango cluster is being disconnected

```
1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut Down
Enter your request: 5

Shutting Down...

C:\dollar\master\mkango>
```

Server

Source Code

Client.py

```
import socket
import threading
import platform
import mimetypes
import os
import sys
import time
from pathlib import Path


class MyException(Exception):
    pass


class Client(object):
    def __init__(self, serverhost='localhost', V='P2P-CI/1.0', DIR='rfc'):
        self.SERVER_HOST = serverhost
        self.SERVER_PORT = 7734
        self.V = V
        self.DIR = 'rfc'  # file directory
        Path(self.DIR).mkdir(exist_ok=True)

        self.UPLOAD_PORT = None
        self.shareable = True

    def start(self):
        # connect to server
        print('Connecting to the server %s:%s' %
              (self.SERVER_HOST, self.SERVER_PORT))
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        try:
            self.server.connect((self.SERVER_HOST, self.SERVER_PORT))
        except Exception:
            print('Server Not Available.')
            return

        print('Connected')
        # upload
        uploader_process = threading.Thread(target=self.init_upload)
        uploader_process.start()
        while self.UPLOAD_PORT is None:
            # wait until upload port is initialized
            pass
        print('Listening on the upload port %s' % self.UPLOAD_PORT)

        # interactive shell
        self.cli()

    def cli(self):
        command_dict = {'1': self.add,
                        '2': self.lookup,
                        '3': self.listall,
                        '4': self.pre_download,
                        '5': self.shutdown}
        while True:
            try:
                req = input(
                    '\n1: Add, 2: Look Up, 3: List All, 4: Download, 5: Shut
Down\nEnter your request: ')
                command_dict.setdefault(req, self.invalid_input)()
            except MyException as e:
                print(e)
            except Exception:
                print('System Error.')
            except BaseException:
                self.shutdown()
```

```
    def init_upload(self):
        # listen upload port
        self.uploader = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.uploader.bind(('', 0))
        self.UPLOAD_PORT = self.uploader.getsockname()[1]
        self.uploader.listen(5)

        while self.shareable:
            requester, addr = self.uploader.accept()
            handler = threading.Thread(
                target=self.handle_upload, args=(requester, addr))
            handler.start()
        self.uploader.close()

    def handle_upload(self, soc, addr):
        header = soc.recv(1024).decode().splitlines()
        try:
            version = header[0].split()[-1]
            num = header[0].split()[-2]
            method = header[0].split()[0]
            path = '%s/rfc%s.txt' % (self.DIR, num)
            if version != self.V:
                soc.sendall(str.encode(
                    self.V + ' 505 P2P-CI Version Not Supported\n'))
            elif not Path(path).is_file():
                soc.sendall(str.encode(self.V + ' 404 Not Found\n'))
            elif method == 'GET':
                header = self.V + ' 200 OK\n'
                header += 'Data: %s\n' % (time.strftime(
                    "%a, %d %b %Y %H:%M:%S GMT", time.gmtime()))
                header += 'OS: %s\n' % (platform.platform())
                header += 'Last-Modified: %s\n' % (time.strftime(
                    "%a, %d %b %Y %H:%M:%S GMT",
time.gmtime(os.path.getmtime(path))))
                header += 'Content-Length: %s\n' % (os.path.getsize(path))
```

```
            header += 'Content-Type: %s\n' % (
                mimetypes.MimeTypes().guess_type(path)[0])
            soc.sendall(header.encode())
            # Uploading
            try:
                print('\nUploading...')

                send_length = 0
                with open(path, 'r') as file:
                    to_send = file.read(1024)
                    while to_send:
                        send_length += len(to_send.encode())
                        soc.sendall(to_send.encode())
                        to_send = file.read(1024)
            except Exception:
                raise MyException('Uploading Failed')
            # total_length = int(os.path.getsize(path))
            # print('send: %s | total: %s' % (send_length, total_length))
            # if send_length < total_length:
            #     raise MyException('Uploading Failed')
            print('Uploading Completed.')
            # Restore CLI
            print(
                '\n1: Add, 2: Look Up, 3: List All, 4: Download\nEnter your request:
')
        else:
            raise MyException('Bad Request.')
    except Exception:
        soc.sendall(str.encode(self.V + '  400 Bad Request\n'))
    finally:
        soc.close()

def add(self, num=None, title=None):
    if not num:
        num = input('Enter the RFC number: ')
        if not num.isdigit():
```

```
            raise MyException('Invalid Input.')
        title = input('Enter the RFC title: ')
    file = Path('%s/rfc%s.txt' % (self.DIR, num))
    print(file)
    if not file.is_file():
        raise MyException('File Not Exit!')
    msg = 'ADD RFC %s %s\n' % (num, self.V)
    msg += 'Host: %s\n' % socket.gethostname()
    msg += 'PORT: %s\n' % self.UPLOAD_PORT
    msg += 'Title: %s\n' % title
    self.server.sendall(msg.encode())
    res = self.server.recv(1024).decode()
    print('Recieve response: \n%s' % res)

def lookup(self):
    num = input('Enter the RFC number: ')
    title = input('Enter the RFC title(optional): ')
    msg = 'LOOKUP RFC %s %s\n' % (num, self.V)
    msg += 'Host: %s\n' % socket.gethostname()
    msg += 'PORT: %s\n' % self.UPLOAD_PORT
    msg += 'Title: %s\n' % title
    self.server.sendall(msg.encode())
    res = self.server.recv(1024).decode()
    print('Recieve response: \n%s' % res)

def listall(self):
    l1 = 'LIST ALL %s\n' % self.V
    l2 = 'Host: %s\n' % socket.gethostname()
    l3 = 'PORT: %s\n' % self.UPLOAD_PORT
    msg = l1 + l2 + l3
    self.server.sendall(msg.encode())
    res = self.server.recv(1024).decode()
    print('Recieve response: \n%s' % res)

def pre_download(self):
    num = input('Enter the RFC number: ')
```

```
        msg = 'LOOKUP RFC %s %s\n' % (num, self.V)
        msg += 'Host: %s\n' % socket.gethostname()
        msg += 'PORT: %s\n' % self.UPLOAD_PORT
        msg += 'Title: Unkown\n'
        self.server.sendall(msg.encode())
        lines = self.server.recv(1024).decode().splitlines()
        if lines[0].split()[1] == '200':
            # Choose a peer
            print('Available peers: ')
            for i, line in enumerate(lines[1:]):
                line = line.split()
                print('%s: %s:%s' % (i + 1, line[-2], line[-1]))

            try:
                idx = int(input('Choose one peer to download: '))
                title = lines[idx].rsplit(None, 2)[0].split(None, 2)[-1]
                peer_host = lines[idx].split()[-2]
                peer_port = int(lines[idx].split()[-1])
            except Exception:
                raise MyException('Invalid Input.')
            # exclude self
            if((peer_host, peer_port) == (socket.gethostname(),
self.UPLOAD_PORT)):
                raise MyException('Do not choose yourself.')
            # send get request
            self.download(num, title, peer_host, peer_port)
        elif lines[0].split()[1] == '400':
            raise MyException('Invalid Input.')
        elif lines[0].split()[1] == '404':
            raise MyException('File Not Available.')
        elif lines[0].split()[1] == '500':
            raise MyException('Version Not Supported.')

    def download(self, num, title, peer_host, peer_port):
        try:
            # make connnection
```

```
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # connect_ex return errors
        if soc.connect_ex((peer_host, peer_port)):
            # print('Try Local Network...')
            # if soc.connect_ex(('localhost', peer_port)):
            raise MyException('Peer Not Available')
        # make request
        msg = 'GET RFC %s %s\n' % (num, self.V)
        msg += 'Host: %s\n' % socket.gethostname()
        msg += 'OS: %s\n' % platform.platform()
        soc.sendall(msg.encode())

        # Downloading

        header = soc.recv(1024).decode()
        print('Recieve response header: \n%s' % header)
        header = header.splitlines()
        if header[0].split()[-2] == '200':
            path = '%s/rfc%s.txt' % (self.DIR, num)
            print('Downloading...')
            try:
                with open(path, 'w') as file:
                    content = soc.recv(1024)
                    while content:
                        file.write(content.decode())
                        content = soc.recv(1024)
            except Exception:
                raise MyException('Downloading Failed')

            total_length = int(header[4].split()[1])
            # print('write: %s | total: %s' % (os.path.getsize(path), total_length))

            if os.path.getsize(path) < total_length:
                raise MyException('Downloading Failed')

            print('Downloading Completed.')
```

```
                # Share file, send ADD request
                print('Sending ADD request to share...')
                if self.shareable:
                    self.add(num, title)
            elif header[0].split()[1] == '400':
                raise MyException('Invalid Input.')
            elif header[0].split()[1] == '404':
                raise MyException('File Not Available.')
            elif header[0].split()[1] == '500':
                raise MyException('Version Not Supported.')
        finally:
            soc.close()
            # Restore CLI
          #  print('\n1: Add, 2: Look Up, 3: List All, 4: Download\nEnter your
request: ')

    def invalid_input(self):
        raise MyException('Invalid Input.')

    def shutdown(self):
        print('\nShutting Down...')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)


if __name__ == '__main__':
    if len(sys.argv) == 2:
        client = Client(sys.argv[1])
    else:
        client = Client()
    client.start()
```

Server.py

```
import socket
import threading
import os
import sys
from collections import defaultdict



class Server(object):
    def __init__(self, HOST='', PORT=7734, V='P2P-CI/1.0'):
        self.HOST = HOST
        self.PORT = PORT
        self.V = V
        # element: {(host,port), set[rfc #]}
        self.peers = defaultdict(set)
        # element: {RFC #, (title, set[(host, port)])}
        self.rfcs = {}
        self.lock = threading.Lock()

    # start listenning
    def start(self):
        try:
            self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.s.bind((self.HOST, self.PORT))
            self.s.listen(5)
            print('Server %s is listening on port %s' %
                  (self.V, self.PORT))

            while True:
                soc, addr = self.s.accept()
                print('%s:%s connected' % (addr[0], addr[1]))
```

```
            thread = threading.Thread(
                target=self.handler, args=(soc, addr))
            thread.start()
        except KeyboardInterrupt:
            print('\nShutting down the server..\nGood Bye!')
            try:
                sys.exit(0)
            except SystemExit:
                os._exit(0)

    # connect with a client
    def handler(self, soc, addr):
        # keep recieve request from client
        host = None
        port = None
        while True:
            try:
                req = soc.recv(1024).decode()
                print('Recieve request:\n%s' % req)
                lines = req.splitlines()
                version = lines[0].split()[-1]
                if version != self.V:
                    soc.sendall(str.encode(
                        self.V + ' 505 P2P-CI Version Not Supported\n'))
                else:
                    method = lines[0].split()[0]
                    if method == 'ADD':
                        host = lines[1].split(None, 1)[1]
                        port = int(lines[2].split(None, 1)[1])
                        num = int(lines[0].split()[-2])
```

```
                title = lines[3].split(None, 1)[1]
                self.addRecord(soc, (host, port), num, title)
            elif method == 'LOOKUP':
                num = int(lines[0].split()[-2])
                self.getPeersOfRfc(soc, num)
            elif method == 'LIST':
                self.getAllRecords(soc)
            else:
                raise AttributeError('Method Not Match')
        except ConnectionError:
            print('%s:%s left' % (addr[0], addr[1]))
            # Clean data if necessary
            if host and port:
                self.clear(host,port)
            soc.close()
            break
        except BaseException:
            try:
                soc.sendall(str.encode(self.V + '  400 Bad Request\n'))
            except ConnectionError:
                print('%s:%s left' % (addr[0], addr[1]))
                # Clean data if necessary
                if host and port:
                    self.clear(host,port)
                soc.close()
                break

    def clear(self, host, port):
        self.lock.acquire()
        nums = self.peers[(host, port)]
```

```
        for num in nums:
            self.rfcs[num][1].discard((host, port))
        if not self.rfcs[num][1]:
            self.rfcs.pop(num, None)
        self.peers.pop((host, port), None)
        self.lock.release()

    def addRecord(self, soc, peer, num, title):
        self.lock.acquire()
        try:
            self.peers[peer].add(num)
            self.rfcs.setdefault(num, (title, set()))[1].add(peer)
        finally:
            self.lock.release()
        # print(self.rfcs)
        # print(self.peers)
        header = self.V + ' 200 OK\n'
        header += 'RFC %s %s %s %s\n' % (num,
                                         self.rfcs[num][0], peer[0], peer[1])
        soc.sendall(str.encode(header))

    def getPeersOfRfc(self, soc, num):
        self.lock.acquire()
        try:
            if num not in self.rfcs:
                header = self.V + ' 404 Not Found\n'
            else:
                header = self.V + ' 200 OK\n'
                title = self.rfcs[num][0]
                for peer in self.rfcs[num][1]:
```

```
                header += 'RFC %s %s %s %s\n' % (num,
                                                 title, peer[0], peer[1])
        finally:
            self.lock.release()
        soc.sendall(str.encode(header))

    def getAllRecords(self, soc):
        self.lock.acquire()
        try:
            if not self.rfcs:
                header = self.V + ' 404 Not Found\n'
            else:
                header = self.V + ' 200 OK\n'
                for num in self.rfcs:
                    title = self.rfcs[num][0]
                    for peer in self.rfcs[num][1]:
                        header += 'RFC %s %s %s %s\n' % (num,
                                                         title, peer[0], peer[1])
        finally:
            self.lock.release()
        soc.sendall(str.encode(header))


if __name__ == '__main__':
    s = Server()

    s.start()
```