Advance Operating System Lab (CSD-416)
National Institute of Technology, Hamirpur

ADVANCE OPERATING SYSTEM LAB (CSD-416) ASSIGNMENT 2

Implement RPC mechanism for a file transfer across a network

AKSHAT RAJ VANSH (185520)

December 11, 2021

Computer Science Department
National Institute of Technology, Hamirpur

Contents

1 RPC mechanism for a file transfer across a network

1.1 Code - Server

```python
import socket as socket
import _thread
import threading
import random
import os
import tqdm
from time import sleep
import time
from client import Client


SEPARATOR = "<SEPARATOR>"
BUFFER_SIZE = 1024 * 5  # 4KB


class Server:
    def __init__(self, port, host=""):
        self.host = host
        self.port = port
        self.files = {}
        self.connection = []

        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def configure(self):
        try:
            self.server.bind((self.host, self.port))
            print("Server binded to port", self.port)
            self.server.listen(5)
            print("Server is listening")
        except Exception as e:
            print(e)

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def send_file(filename, host, port):
        # get the file size
        filesize = os.path.getsize(filename)
        # create the client socket
        s = socket.socket()
        print(f"[+] Connecting to {host}:{port}")
        s.connect((host, port))
        print("[+] Connected.")

        # send the filename and filesize
        s.send(f"{filename}{SEPARATOR}{filesize}".encode())

        # start sending the file
        progress = tqdm.tqdm(range(
            filesize), f"Sending {filename}", unit="B", unit_scale=True, unit_divisor=1024)
```

```
55          with open(filename, "rb") as f:
56              while True:
57                  # read the bytes from the file
58                  bytes_read = f.read(BUFFER_SIZE)
59                  if not bytes_read:
60                      # file transmitting is done
61                      break
62                  # we use sendall to assure transimission in
63                  # busy networks
64                  s.sendall(bytes_read)
65                  # update the progress bar
66                  progress.update(len(bytes_read))
67
68          # close the socket
69          s.close()
70
71      def find(self, filename):
72          return self.files[filename]
73
74      def connect(self, sender, reciever):
75          file = self.rec_file(sender)
76          print(file)
77          reciever.sendall(self.encode("FILE({})".format(file)))
78
79      def rec_file(self, sender):
80          print("[+] Waiting for file...")
81          bytes_read = self.decode(sender.recv(BUFFER_SIZE))
82          print("[+] File received.")
83          return bytes_read
84
85      def listen(self, client, client_addr):
86          while True:
87              data = client.recv(BUFFER_SIZE)
88              data = self.decode(data)
89              message = data[:data.find('(')]
90              if(message == "F"):
91                  file = data[data.find('(')+1:data.find(')')]
92                  for f in file.split("\n"):
93                      print(f)
94                      self.files[f] = client
95              if message == "REQUEST":
96                  filename = data[data.find('(')+1:data.find(')')]
97                  filesize = os.path.getsize(filename)
98                  sender = self.find(filename)
99                  sender.send(self.encode("SEND({})".format(filename)))
100                 _thread.start_new_thread(self.connect, (sender, client))
101
102     def threaded(self, client, client_addr):
103         _thread.start_new_thread(self.listen, (client, client_addr))
104         while True:
105             continue
106
107     def start(self):
108         self.configure()
109         while True:
110             client, client_addr = self.server.accept()
111             self.connection.append(client)
112             print('Connected to :', client_addr[0], ':', client_addr[1])
```

```
113
114             _thread.start_new_thread(
115                 self.threaded, (client, client_addr[1]))
116
117
118 if __name__ == '__main__':
119     server = Server(1237)
120     server.start()
```

1.2 Code - Client

```
import socket
import json
import _thread
import time
import random
import os
import tqdm
import threading
import os
import subprocess

BUFFER_SIZE = 5120
SEPARATOR = "<SEPARATOR>"

class Error:
    commandInputError = Exception("Please enter correct command")
    portInputError = Exception("Please enter correct port number")
    controllerError = Exception("Controller Error. Try After Sometime")
    createRoomError = Exception("Error in creating the room")


class Client:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.connections = []
        self.fileName=""
        self.weight = ""

    def createSocket(self, port):
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((self.host, port))
        return client

    def decode(self, value):
        return value.decode('ascii')

    def encode(self, value):
        return value.encode('ascii')

    def rec_file(self,file,):
        filesize=BUFFER_SIZE
        progress = tqdm.tqdm(range(filesize), f"Receiving {self.fileName}", unit="B", unit_scale=Tru
        with open(self.fileName, "wb") as f:
                f.write(self.encode(file))

            # update the progrclient, client_address bar
        # for i in range(filesize):
        #     progress.update(i)
        progress.update(filesize)



    def listen(self, client):
        while True:
            data = client.recv(BUFFER_SIZE)
```

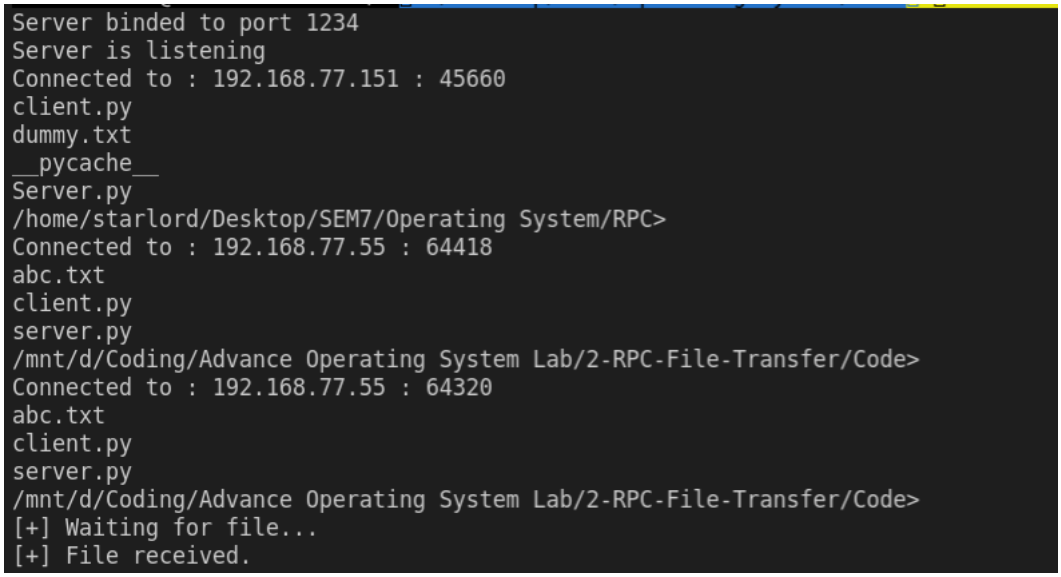
```
            data = self.decode(data)
            message = data[:data.find('(')]
            if(message == "SEND"):
                fileName = data[data.find('(')+1:data.find(')')]
                print(fileName)
                self.send_file(fileName,client)
            if(message=="FILE"):
                file = data[data.find('(')+1:data.find(')')]
                print(file)
                self.rec_file(file)

        client.close()
        exit(0)

    def send_file(self,filename,client):
        # get the file size
        filesize = os.path.getsize(filename)
        # create the client socket
        # start sending the file
        progress = tqdm.tqdm(range(
            filesize), f"Sending {filename}", unit="B", unit_scale=True, unit_divisor=1024)
        with open(filename, "rb") as f:
            while True:
                # read the bytes from the file
                bytes_read = f.read(BUFFER_SIZE)
                if not bytes_read:
                    # file transmitting is done
                    break
                # we use sendall to assure transimission in
                # busy networks
                client.sendall(bytes_read)
                # update the progress bar
                progress.update(len(bytes_read))

    def send(self,client):
        while True:
            message = input("")
            if(message[:message.find("(")]=="REQUEST"):
                self.fileName = message[message.find("(")+1:message.find(")")]
                client.send(self.encode(message))



        client.send(self.encode(message))
        client.close()
        exit(0)
    def start(self):
        client = self.createSocket(self.port)
        cmd = subprocess.Popen("ls",
                               shell=True, stdout=subprocess.PIPE,
                               stdin=subprocess.PIPE, stderr=subprocess.PIPE)
        output_byte = cmd.stdout.read() + cmd.stderr.read()
        output_str = str(output_byte,"utf-8")
        currentWD = os.getcwd() + "> "
        client.send(self.encode("F({})".format(output_str + currentWD)))
    # _thread.start_new_thread(self.send, (client,))
        _thread.start_new_thread(self.send, (client,))
        _thread.start_new_thread(self.listen, (client,))
```

```
115         while True:
116             continue
117 
118 if __name__ == '__main__':
119     client = Client('192.168.77.151',1234)
120     client.start()
```

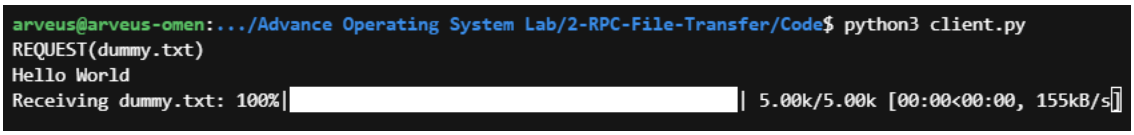
1.3 Output

RPC mechanism for a file transfer across a network Outputs

Server Side



Client - Sender



Client - Receiver