Assignment Report

Huanrui Zhang, z5185099

note.

1. python version: 3.7.1
2. press ctrl+c twice to kill a peer, please
3. demo link: https://youtu.be/kQ6_WuDpSrc

1 Introduction

In my program, there are 1 main thread and 3 subthreads. The function of the main thread is sending ping request and receiving ping response by UDP. The first subthread receives the ping request or the file data by UDP. The second subthread sends the file request if get 'request file' command and makes peer departure if receive 'quit' command by TCP. The third subthread is the TCP server which receives the file request. If the file is not stored in this peer, it will transfer the message to the successor. Otherwise, it will send the file to the peer which sends the 'request file' command by UDP. Meanwhile, it deals with peer departure and kill peer events.

2 Ping

The interval between pings is 10 second. The timeout for receiving ping response is 2 second. The number of lost packets before assuming the peer is killed is 3. The corrsponding code shows below. Main thread: time.sleep(10) is the interval between pings. It also shows that I use UDP for ping.

```
# send the ping request every 10s
while True:
    # work_flag(0: already depart)
    if work_flag == 1:
        m_1 = ' '.join(('first_successive_id', str(own_id), str(ping_seq)))
        m_2 = ' '.join(('second_successive_id', str(own_id), str(ping_seq)))
        clientSocket_first.sendto(m_1.encode(), ('127.0.0.1', 50000 + first_successive_id))
        receive_ping_response_first()
        clientSocket_second.sendto(m_2.encode(), ('127.0.0.1', 50000 + second_successive_id))
        receive_ping_response_second()
        ping_seq += 1
        time.sleep(10)
    else:
        sys.exit()
```

If the value of first-alive-flag is bigger than 2, then the peer successor is killed.

```
 try:
    data, (address, _) = clientSocket_first.recvfrom(1024)
    if data and int(data.decode().split()[0]) == first_successive_id:
        print(f'A ping response message was received from Peer {first_successive_id}')
        first_alive_flag = 0
except OSError:
    first_alive_flag += 1
    if first_alive_flag >= 2:
        print(f'Peer {first_successive_id} is no longer alive.')
        first_successive_id = second_successive_id
        print(f'My first successor is now peer {first_successive_id}.')
        Kill_clientSocket = socket(AF_INET, SOCK_STREAM)
        Kill_clientSocket.connect(('127.0.0.1', 50000 + first_successive_id))
```

```
            message = ' '.join((str(own_id), 'first', 'kill'))
            Kill_clientSocket.send(message.encode())
            second_successive_id = int(Kill_clientSocket.recv(1024).decode())
            print(f'My second successor is now peer {second_successive_id}.')
            Kill_clientSocket.close()
```

I have tested my program several times and find this time is reasonable. Ping request does not make the link crowded, but peers can know when the successor is not alive in a short time.

3 File request and Departure

I use TCP for the sending file request/response message and sending the departure message.

```
    # request a file (TCP)
    def request_file():
        global predecessor_id
        global work_flag
        while True:
            command = input()
            command = command.split()
            # file request/response message
            if len(command) == 2 and command[0] == 'request' and command[1].isdigit():
                filename = int(command[1])
                if filename >= 0 and filename < 10000:
                    print(f'File request message for {filename} has been sent to my successor.')
                    # file client
                    TCP_clientSocket = socket(AF_INET, SOCK_STREAM)
                    TCP_clientSocket.connect(('127.0.0.1', 50000 + first_successive_id))
                    file_request = ' '.join((str(own_id), str(own_id), str(filename)))
                    TCP_clientSocket.send(file_request.encode())
                    TCP_clientSocket.close()
            # departure message
            elif command and command[0] == 'quit':
                TCP_clientSocket = socket(AF_INET, SOCK_STREAM)
                TCP_clientSocket.connect(('127.0.0.1', 50000 + predecessor_id[0]))
                quit_message = ' '.join((str(first_successive_id), str(second_successive_id), 'quit'))
                TCP_clientSocket.send(quit_message.encode())
                TCP_clientSocket.close()
                TCP_clientSocket = socket(AF_INET, SOCK_STREAM)
                TCP_clientSocket.connect(('127.0.0.1', 50000 + predecessor_id[1]))
                quit_message = ' '.join((str(predecessor_id[0]), str(first_successive_id), 'quit'))
                TCP_clientSocket.send(quit_message.encode())
                TCP_clientSocket.close()
                print(f'Peer {own_id} will depart from the network.')
                work_flag = 0
```

4 possible improvements

For detecting the killing peer event, I think that the peer can detect whether its successor is not alive only. If the successor is not alive, the current peer updates own first and second successor table first and then tells predecessor to update the predecessor's first and second successor table. The current can know which peer is its first predecessor by the ping request it received. By doing this, the program gets rid of the situation that current peer ask its successor what is the second successor is, but the first successor have not update its new successor.