COMP 9331

Platform: python3
Version: 3.6.2
Screencast Demo: https://youtu.be/8XM0h5zgMcE
Student id: 5166715 Name: Zeyuan Ma

Choosing python as the code for this assignment, because of the code of python is terse, clear and easy to understand. In addition, the code of python can reveal the concept of socket programing.

There three main threads in the program, one thread hold the UDP for ping successors, one hold the TCP client and the last one hold the TCP server. The program is designed step by step and the report will explain the program step by step.(See appendix,figure1)

The step 1 is initialization, the step pass 3 argument to the cdht.py by command. The first is the identity of the peer and the other two are the identity of the first and second successor.

For the step 2, Ping successors, because of UDP do not need a build a steadily 'link' between server and client, a function is built to achieve the client and server, from appendix, figure 2, it can be seen that a UDP function is built and the socket is bound by an address as the identity of peer. Therefore, it can send and receive message with clear identity. Then, the function is hold by a UDP thread. (See appendix, figure 3)

For the step 3, requesting a file, there are two thread, one for TCP client, another for TCP server.
The client create a socket to connect the successor of the current peer. The client also convert the input string of user into the hash number of the file and put it into a tuple with the identity of current peer, then send it to the server of successor, as string. (See appendix, figure 4)
The server create sockets for accept the connect request and receive the file request message. The server will determine whether the peer be responsible for the request file. If 'yes', create a socket to the peer which request. If 'no', create a socket to connect its successor and send the request message to the successor. (See appendix, figure 5)

For the step 4, peer departure, The TCP client will judge what the user input, if it is "request a file", the step 3 will be executed, if it is "quit", create a socket to connect its successor to send quit message and close the peer. The message contain the identity and two successor of the quit peer and quit message. (See appendix, figure 4)
The server will also determine whether the message is "request" or "quit". If "request", executing step 3, if 'quit', judge that the current peer is the predecessor of quit peer. If yes, changing the successor of the current peer to the successor of quit peer. (See

appendix, figure 5)

Appendix

Figure 1

```
def main():

    TCP_thread_s = threading.Thread(target=thread_job_s)
    UDP_thread = threading.Thread(target=UDP_thread_job)
    TCP_thread_c = threading.Thread(target=thread_job_c)


    TCP_thread_s.start()
    TCP_thread_c.start()
    UDP_thread.start()


main()
```

Figure 2

```
def UDP_thread_function(arg1,arg2,arg3,addr):
    s = socket(AF_INET, SOCK_DGRAM)
    s.bind(addr)
    while True:

        message = b"Ping message"
        des_port1 = 50000 + arg2
        des_port2 = 50000 + arg3
        des_addr1 = ("",des_port1)
        des_addr2 = ("",des_port2)
        s.sendto(message,des_addr1)
        s.sendto(message,des_addr2)

        data, server = s.recvfrom(1024)
        pe_number = server[1]-50000
        if data == b"Ping message":
            respond = b"Respond message"
            s.sendto(respond,server)
            print(f'A ping request message was received from Peer {pe_number}')
        elif data == b"Respond message":
            print(f'A ping  response message was received from Peer {pe_number}')
        time.sleep(3)
```

Figure 3

```
def UDP_thread_job():
    global arg1
    global arg2
    global arg3

    N_port = 50000 + arg1

    N_addr =("",port)

    UDP_thread_function(arg1,arg2,arg3,N_addr)
```

Figure 4

```
def thread_job_c():
    global arg1
    global arg2
    global arg3
    global addr
    global U_addr


    TCP_des_port = 50000+arg2
    TCP_des_addr = ("127.0.0.1",TCP_des_port)

    TCP_c_port = 60000+arg1
    TCP_c_addr = ("127.0.0.1",TCP_c_port)




    while True:
        sentence = input('Which file request or quit:\n')
        if sentence == "quit":
            c_s = socket(AF_INET,SOCK_STREAM)
            c_s.bind(TCP_c_addr)
            c_s.connect(TCP_des_addr)
            fi_se_tuple = ("quit",arg1,arg2,arg3)
            fi_se = str(fi_se_tuple)
            c_s.send(fi_se.encode('utf-8'))
            os._exit(0)
        else:
            c_s = socket(AF_INET,SOCK_STREAM)
            c_s.bind(TCP_c_addr)
            c_s.connect(TCP_des_addr)
            file_value = request_file_value(sentence)
            sentence_value = file_value%256
            tuple_hash = (arg1,sentence_value,file_value)
            str_hash = str(tuple_hash)
            c_s.send(str_hash.encode('utf-8'))
            TCP_respond = c_s.recv(1024)
```

Figure 5

```
def thread_job_s():
    global arg1
    global arg2
    global arg3
    global addr


    s_s = socket(AF_INET,SOCK_STREAM)
    s_s.bind(addr)
    s_s.listen(5)
    print("The server is ready to receive")
```

```
while True:
    connect_s,sou_addr = s_s.accept()
    sou_peer = sou_addr[1]-60000
    sentence = connect_s.recv(1024)
    sentence = sentence.decode('utf-8')
    sentence_tuple = tuple(eval(sentence))
    if sentence_tuple[0] == "quit":
        if arg2 == sentence_tuple[1]:
            de_peer = sentence_tuple[1]
            print(f"Peer {de_peer} will depart from the network.\n")
            arg2 = sentence_tuple[2]
            print(f"My first successor is now peer {arg2}.\n")
            arg3 = sentence_tuple[3]
            print(f"My second successor is now peer {arg3}.\n")
        elif arg3 == sentence_tuple[1]:
            de_peer = sentence_tuple[1]
            print(f"Peer {de_peer} will depart from the network.\n")
            print(f"My first successor is now peer {arg2}.\n")
            arg3 = sentence_tuple[2]
            print(f"My second successor is now peer {arg3}.\n")
            de_p_f(sentence)
        else:
            de_p_f(sentence)

    else:
        hash_value = sentence_tuple[1]

        if sentence_tuple[0] == 'R':
            file = sentence_tuple[1]
            print(f"Received a response message from peer{sou_peer}, which has the file{file}.")
        else:
            file =  sentence_tuple[2]
            TCP_pe_number = sou_addr[1]-60000

            request_peer = sentence_tuple[0]
            request_port = sentence_tuple[0]+50000


            if TCP_pe_number < hash_value <= arg1 or TCP_pe_number > arg1 and hash_value > TCP_pe_number:
                print(f"File {file} is here")
                new_c_so(request_port,file)
            else:
                print(f"File {file} is not stored here")
                new_c_s(sentence)
```