COMP3331 Assignment

Python version: 2.7
YouTube URL:
https://www.youtube.com/watch?v=Z1Rvj-VgmJQ

Program design

*several auxiliary printouts should show up on terminal. They are:

Successor[1/2] is XX

Parent[1/2] is xx

Timeout!!! (for knowing one of the response doesn't arrive)

Other printouts follow the specification.

Global variable:

A global variable called **flag** is set for all loops inside each peer, it will terminate all loops of this peer when a keyboard interrupt appears or 'quit' command is executed.

A global string called **important** carries all the messages that peer n's server want its own client to send to its successor.
Every time the message in **important** is sent, it will be cleaned up to an empty string

Dictionary **successor** is for storing two successors.

Dictionary **parent** is for storing two parents.

Dictionary **request** is for counting how many request the peer send to a certain successor.

Dictionary **respond** is for counting how many respond the peer receive from a certain successor.

Message design:

FILE_REQUEST([0-9]{4}),([0-9]{1,3}),([0-9]{1,3}),([01])

Request a file, its complete file name is the first 4 digits immediately followed by 'FILE_REQUEST', and then is the hash file name, the is the requesting peer's identity, last one digit is whether it check around the circle(like 15->1).

FIND([0-9]{1,3}),([0-9]{4})

When a peer finds it has the file it will send the message to the requesting peer. This message includes the identity of the peer who has the file and 4 digits file name and requesting peer's identity.

QUIT_INFO

When user types command 'quit', peer N sends QUIT message to its two predecessors. Parent 1 is the peer that its immediate successor is N and parent 2 is the peer that its second successor is N.

QUIT_INFO_TWO is sent to parent 1, since parent 1 needs to know both

successors of N so that it can connect to these new successors.

QUIT_INFO_ONE is for parent 2, parent 2 only need to know N's immediate successor so only one number is sent.

ASK_SUCC

When receiving this header, server sends its immediate successor to the corresponding client.

Run five threads in the main function:

1. **TCP server thread**

   In this thread, server receives all the messages that its predecessor sends to it. According to the header of message, it decides which action it should take. There are three kinds of headers now:

2. **TCP client thread**

   In this thread, there are three sub-threads.

   inputFrame():

   It is for waiting for user's input.

   impoFrame():

   This is for checking whether global string 'important' is filled with new message.

   checkAlive():

   It keeps check whether any of its successor is killed. Once such successor is detected, it will update successor information and make new connection with them.

3. **UDP server thread**

   Keep replying the ping message from its predecessor.

4. **Two UDP client threads**

   Each peer has two successors, so two UDP client threads are started. Each keeps sending ping message to a specific successor.

How the system works:

Begin with user types a command into the terminal.

It should follow one of the standard formats:

1. request [0-9]{4}
2. quit
3. ctrl+c (kill peer)

For request [0-9]{4}:

The filename will send to a function called getFileNum() to check whether it is a valid file name. Then it is calculated into a hash file name < 256. Finally, it is wrapped into a FILE_REQUEST.. message and sent to its immediate successor.

When successor M accepts the FILE_REQUEST message, it grabs the information from it and check whether it has the file. If it has, it will make a FIND message and put it into 'important', client finds that 'important' is filled and is with FIND header, then it set up a connection with the requesting peer and send the

message to it.

For quit:

When getting quit command, the thread invokes two sub-threads, both using tellParent() function to tell its parents its departure. The successors' messages are wrapped in a QUIT_INFO message in this function. It change the global flag to 1, resulting all loop stopped, and then print out it is going to depart.

For one peer killed by user:

When a certain peer is killed by user, it predecessors will still send ping to it, and of course it cannot reply.

In TCP client there is thread called checkAlive(). It will keep tracking the number of request and response from each successor. Once the difference between request and response is greater than a certain number, it will conclude this successor is killed.

Design trade-offs considered and made:

It starts with four main threads so that it looks clean and we don't have worry about the things outsides the thread. Each thread plays a unique role as a server or client for UDP or TCP.

All the actions we have to keep checking are also put into separate thread so that they will nothing get intersected.

This makes it easier to extend implementation of new commands.

Particular circumstances:

Some tips we have to be careful:

a) all command should be input after both UDP and TCP set up, that is two line 'UDP server is ready to receive' and 'the TCP server is ready to receive'. Also, parent should not be 0 (it can be seen on the terminal)
   If one of them fails to set up, please restart.
b) When 'peer N will depart from the network' (also apply to 'Peer N is killed') shows up in the terminal, there should be some messages arrive, since they are sent before it departs or killed.
c) After a peer quit, for the first time the parent peer want to request a file, the parent may have to input file request command twice. (This is rather important!)
d) The normal time of detecting there is a successor killed is about 40 ~ 50s. Two predecessors may not refresh successors at same time, please wait for a little while.

All code is originally written.