Leader :

Julian Saavedra 500449774

Other Members:

Jenson Manalil 500336863

Johnpaul McCaslin 500278413

**DHTServer.java**

A DHTServer object keeps track of the following

* It’s own Server ID
* It’s TCP Port for the other DHT Servers
* It’s own IP Address
* It’s successor and predecessor IP Addresses
* It’s successor and predecessor TCP Welcome Ports (Usually the same as its port, unless configured otherwise)
* A DatagramSocket for P2P Clients
* A HashMap containing the records of P2P Clients and the files they hold

main(String [] args)

* Initializes the port and ID (through user arguments)
* Creates a new DHTServer

DHTServer(int newPort, int ID)

* When called, creates a DHTServer, assigning the port and Server ID from the parameters
* Creates a ServerSocket that waits on port newPort

stringToInt(byte[] IP)

* Used for converting string of IP to an int to store in HashMap (which is in the format of hash(String, int).

intToIP(int IP)

* Does the reverse of stringToInt

requestUnknownIP()

* Grabs the IP address from the successor server, by anticipating an input via the stop method readUTF().
* Returns this IP Address (InetAddress)

waiForPrevServer()

* Waits for a connection on its Welcome Socket
* Store this socket to prevSocket (predecessor socket)
* Creates input and output streams when it receives one

connectToNextServer(String ip, int port)

* Create a socket to connect to IP Address ip and Port port
* Store this socket to nextSocket (successor socket)
* Create input and output streams

run()

* The loop of the program
* Creates a UDP socket and set a timeout for it (10 seconds).
* When timeout is reached, send the its own IP Address to the predecessor (which is going to be catched by the requestUnknownIP() method if it is called, otherwise it will be ignored. This is done through the TCP connection of the servers. The loop will be continued, and the UDP will be recreated and waiting in addition to its timer reset.
* At first it checks if it’s initializing or not, when it is initializing, and its Server ID is 2 to 4, it will wait for a connection (waitForPrevServer() function) on its Welcome Socket, and when a connection is received, create a connection socket, and send a request to the next Server IP (connectToNextServer() function). Store the IP and Port of the successor and predecessor. If it is Server ID 1, it will send a request to Server ID 2 and then wait for a request from Server ID 4. Furthermore this is where we initialize its UDP port.
* When it is done the initializing phase, it will wait for a UDP packet from a client, and this packet will contain a number in the form of a String ranging from 1 to 3 and 5.
* Whenever a UDP packet is received (A request is made), Server will grab its IP Address and Port
* If the packet is “1” (only Server ID 1 will receive this) Server will create 4 InetAddresses. The first will contain its own IP, the next two will contain the predecessor and successor IP. The last will contain the last IP by calling the method called requestUnknownIP(), described earlier. The InetAddress will be converted to a String by concatenating them and the String will be sent the client.
* If the packet is “2” Server will tell the Client that it has received a request, and will wait for a String which is the file name. Once it receives the file name it will convert the Client’s IP Address to an int via the stringToInt() function, and store the content name and IP Address in the hashMap.
* If the packet is “3” Server will tell Client it has received a request and will wait for a String which is the file name. Once it receives the file name, it converts the IP Address to int, then checks the hashMap if there exist a record (Filename, IP Address). If it does, send the IP Address of the client who is responsible for that content to the requesting client. If it does not, tell the client it does not exist (404).
* If the packet is “5” , convert client IP Address to int, iterate to hashMap, then delete anything related to that IP Address. If something is deleted tell client how many files it deleted, if it did not find anything, tell client, it has nothing to delete.

**P2PClient.java**

A P2PClient holds the following

* An array for the IPs of the servers
* An array for the ports of the servers
* A Boolean to determine if this client is a server (listening for TCP connections)
* An array containing a string of contents

hash(String content)

* The hash function
* Adds each letter of the string and into X
* Y = X MOD 4 where Y + 1 is Server ID
* Return Y

sender(String msg, DatagramSocket socket, InetAddress address, int port)

* Turns msg into bytes, create a packet with the parameters address and port
* Send that packet into the DatagramSocket socket
* Used for sending messages to the DHT Servers

waitAndGet(DatagramSocket socket)

* Create a byte[] for the received content
* Create a DatagramPacket for the packet about to be received
* Do socket.receive() which is a stop method
* When a packet is received convert it into a String
* Return this string
* This is used for received messages from the DHT Servers

main()

* Takes in an input, call it sentence
* Creates a DatagramSocket, and get the IP Address of the Server ID = 1 (hard coded)
* Fill in the server ports array (usually they are the same since they are different machines)
* If sentence is 1, client sends a packet = “1” to the IP Address of Server ID 1 and to its UDP port. Wait for a response. When response is received, split that String (the concatenation of IP Addresses) into separate strings and put them into the serverIPs[] array.
* Assuming client initialized (did request “1” first), If the sentence is “2”, ask for the name of the file from user. Once client receives the name, we hash this via hash() function, and determine which server to send to. Once we determined the server, we send the request sentence “2” to that server. We wait for a response via the waitAndGet() method, and once we received a response we send the file name to the server via the sender function. If listening is false, we create a P2PServer() thread that listens to TCP connections at a specific port.
* Assuming client initialized, if the sentence is “3”, we ask for the name of the file from user. Once received, we hash that name and determine the server to send to. Once we determined the server, we send the request “3” and do waitAndGet(). Once we received a response we send the file name to the server. We do another waitAndGet() to get the IP Address of the client who is responsible for the file, once we received the IP address we create a new socket containing the P2P server’s IP Address and the assumed port. We send a request to it, and wait for a response. Once we received a response (which is going to contain the new port to TCP to) the client will create a new socket to that new port, and send the file name. Client will wait for a response, which are the file bytes. We first use a stream from the P2P Server to P2P Client, and then we use another for P2P Client to Desktop. Once the file transfer is complete, we close the sockets used for file transfer.
* Assuming client initialize, if the sentence is “5”, we send the packet 5 to all DHT Servers, which are known from initializing.

**P2PServer.java**

P2PServer holds the following

* It’s listening TCP socket (welcome socket)
* An int (ftpSock) which increments everytime a file transfer has been asked
* An int which contains the port of the socket being requested to
* A thread
* Input and output streams

P2PServer(int socketNum)

* When called, we make ftpSock, socket (both integers) to the parameters socketNum
* We create a ServerSocket(socketNum)

start()

* When thread is null, we create a new one and start it.

run()

* When current thread is running, we check if socket equals to the assumed TCP listener port. This means a client is asking for a FTP. If it is we increment ftpSock, so that we won’t use a TCP port used before to avoid closed connections that are being currently used. We create a connection socket for client trying to connect. We tell the client to connect to the new port by sending it the port int. We create a new P2PServer with the port ftpSock. Which will create a new thread
* When current thread is running and the socket is not the assumed TCP listener port. This means a P2PServer received a FTP request from client; the client received the new port number and sent a new request to the P2PServer (with the incremented port number ftpSock). We then call the method fileTransfer().

fileTransfer()

* We create a new socket for client trying to connect.
* We open up the streams to be used
* We wait for an input from the new socket (this is the file name)
* Once we received the input we concat a “.jpg” to it.
* We first create a FileInputStream to get the file from desktop, we then use a BufferedOutputStream to send that file to the requesting client.
* Once everything is sent, we close the streams used and end the current thread.