Blaine McMahon, Jacob Sword, Nicholas Yameen

Network Design

Project Phase 2

2/14/19

**1. Introduction:**

This code completes Phase 2 of the project by using UDP to send images from a client to a server, and vice versa. The UDP server is run continuously to accept messages from clients. These messages are in the form of 1. A filename of the image the client will send, and 2. The contents of said image. The server then writes the file. After sending this file, the client waits for a response of the same nature from the server and then writes it’s transferred file before exiting. The below flowcharts and explanations explain these procedures in more detail.

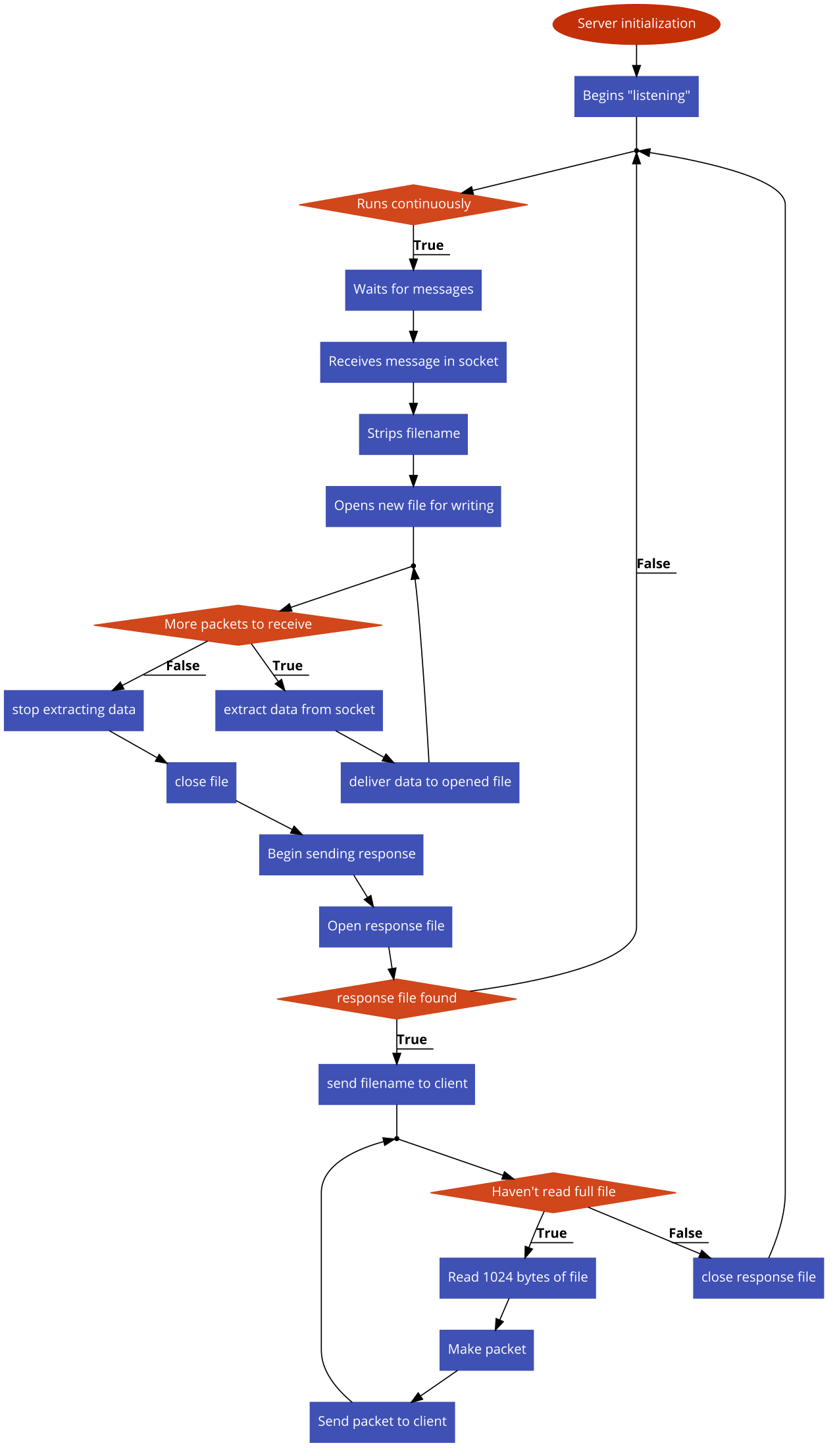
**2. Flowcharts**

Fig 1: Sever Flowchart

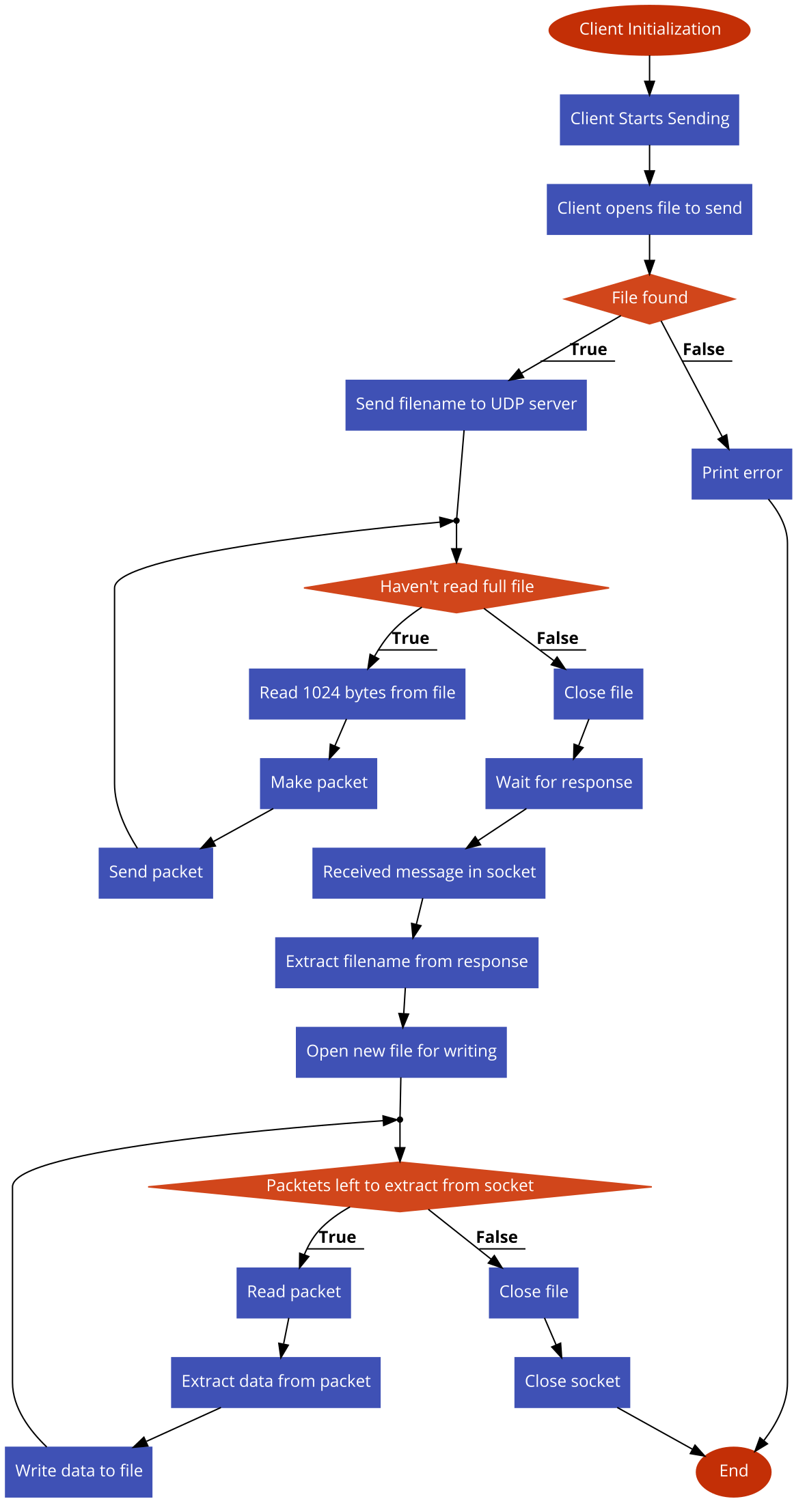


Fig 2: Client Flowchart

**GUI.py**

This file is the main GUI for the program. It uses tkinter to create the GUI. First, the GUI window will open and display three buttons and a place to enter text. The user can either write the path to the file to be sent or leave it blank to use our default image. The user then has to click Start Server first. This will start the *run\_server()* function. This function will update the gui text to show that the server is listening and then call *Server()*. This function initializes a Server object and then runs it in a new thread. Next, the user can click Start Client button. This will call *start\_client()* which updates the gui to show that the client is working, then calls the *Client()* function. This function initializes a client class with a “callback” parameter. This callback is used within the class to call a function when the client’s work is done. In this case, the function changes the GUI message from “Client working…” to “Client finished.” Client then calls the client object’s start\_send process in a new thread. The two scripts are run as described below.

**UDPserver.py**

In the UDP server file, there is a class called UDPserver. The init function will first either use the file path given by the user at the start or use the default if not specified by the user. It will then open a UDP socket and bind to it, and make it an instance variable.

When called, the listen() function will continually run, where it will listen for incoming connections. The sever will first receive the filename being sent from the client. It will take that name and open the file for writing binary. It will then preform a *rdt\_rcv()* where it will continually receive packets from the client and write them to the file until there are no packets left. Once the full image is received it will then close the file and send an image back to the client. Messages will be printed to the console through this to show the status. All messages printed from the server are run through “self.print()” to prefix it with the “Server:” string for easier reading in the terminal.

The *send\_img()* function will first open the file specified or the default one for reading binary. It will then send the image name to the client. Finally, it will send the image using the *rdt\_send()* function.

**UDPclient.py**

The UDPclient has a class which defines four functions. First, the client will initialize similar to the UDP server. It will connect to the server on the UDP socket, which is made into an instance variable. Then it will go into *start\_send()* function. It will first call *send\_img()* function to send the server an image.

The *send\_img()* function will first either use the path given by the user or the default one. If there is an error finding the given filename, it will be caught above and the client will stop execution. If the file is successfully found, it will open that image for reading binary. The client will send the server the image name. Then it preforms RDT send by calling the *rdt\_send()* function. Once it has finished it will close that file.

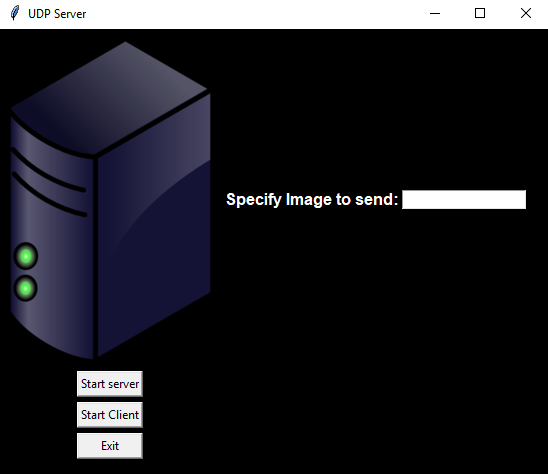
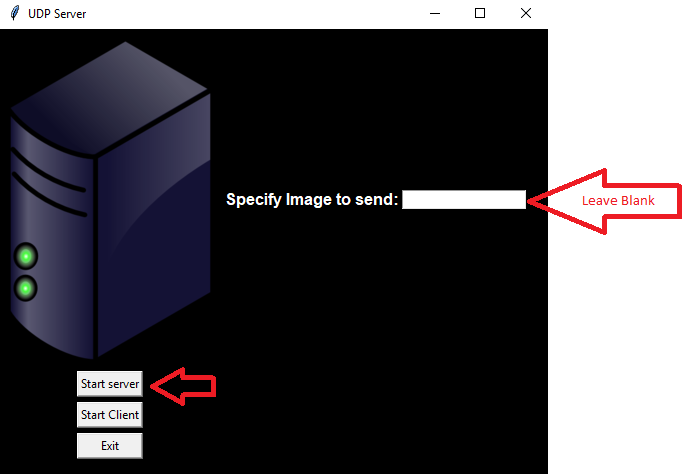
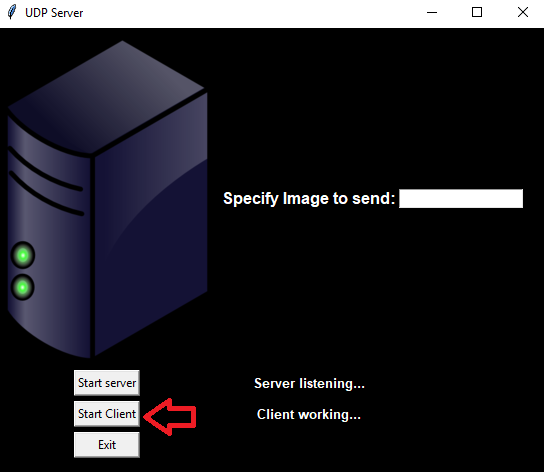
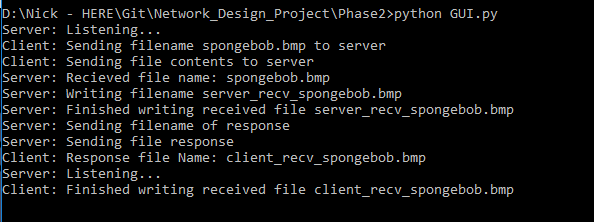
Next, the *start\_send()* will call the *wait\_and\_receive()* function. Here the client will receive the response image name from the server. It will open that file for writing binary. After which it will call *rdt\_rcv()*  to receive all the packets of the image. After which, the file will close, and the callback function will be called. This function just prints it has finished. The client will then close the socket and be finished. The client also uses a self.print function to prepend “Client:” to all print strings.

**rtd.py**

In the file, there are six functions defined. The server and client will use these functions to send RTD 1.0 over a socket. The sender will use *rdt\_send()* function. This function will take in a file, endpoint, and the socket. It will make packets using the *make\_pkt()* function by reading 1024 bytes at a time. It will go into a while loop, which will run while there is a packet to be sent. The *udt\_send()­­* function will send the packet to the endpoint over the socket. It returns the number of bytes sent, then make a new packet. It will continue to send packets until the whole file has been sent.

The *rdt\_rcv()* function will receive a packet. It will be called with the filename that is being received, the file, and the socket. It will go into a while True loop where it will first call *extract()* function. *Extract()* will wait for thread to be ready I/O with the select.select() line. If it is not, it will timeout, which means there is no data to receive on the socket. If the thread is ready, it will receive 1024 bytes on the socket and return the packet. In the *rdt\_rcv()* will then call *deliver\_data()* which will take the data and file. This function will write the file. The receive function will continue to extract and write packets until it has reached the end. At that time, the file will be closed and will be broken out of the while loop.

**How to Run**

1. Make sure all files laid out in the ReadMe.txt are present.
2. Issue the command “python GUI.py” on the command line. The following GUI will appear.  
   
3. Leave file to send blank to use default or specify path to your own picture. Click server  
   
4. Again, leave image blank to use default, or specify a different image. Then click client.  
   
5. You will see messages on the GUI, as seen in the image above. In the terminal messages will appear of what is happening as seen below.  
   
6. New files will be prefixed with “client\_recv\_{name}” and “server\_recv{name}” depending on which entity they were received by.
7. Click client again to re-run the client or click exit to stop.