Stevens Institute of Technology
Department of Computer Science

CS 549: Distributed Systems and Cloud Computing
Homework 1 – RMI and Sockets

Student: Gurpreet Singh
Email: gsingh15@stevens.edu

1. Connecting the client to the server

File: Client.java

```
/*
 * TODO: Get a server proxy.
 */

IServer server = null;
Registry registry = LocateRegistry.getRegistry(serverMachine, serverPort);
IServerFactory serverFactory = (IServerFactory) registry.lookup(serverName);
server = serverFactory.createServer();
```

The code above creates a remote connection with the server using the server machine, server name, and server port. The client first obtains the stub for the registry by invoking the static LocateRegistry.getRegistry method with the serverMachine and serverPort specified. Next, the client invokes the remote method lookup on the registry stub to obtain the stub for the remote object from the server's registry. Finally, the client invokes the createServer method to create connection with the server.

2. Get the data from the server to the client in PASSIVE and ACTIVE modes.

File: Client.java

```
private static class GetThread implements Runnable {
    /*
     * This client-side thread runs when the server is active mode and a
     * file download is initiated. This thread listens for a connection
     * request from the server. The client-side server socket (...)
     * should have been created when the port command put the server in
     * active mode.
     */
    private ServerSocket dataChan = null;
    private FileOutputStream file = null;

    public GetThread(ServerSocket s, FileOutputStream f) {
        dataChan = s;
        file = f;
    }

    public void run() {
        try {
            /*
             * TODO: Complete this thread.
             */
            msgln("5. Client waiting for connection from the server");
            msgln("Addr " + dataChan.getLocalSocketAddress() + " " + dataChan.getInetAddress());
            Socket xfer = dataChan.accept();

            msgln("7. Client waiting for data from the server");
            InputStream in = xfer.getInputStream();
            byte[] buf = new byte[BUFFER_SIZE];
            int nbytes = in.read(buf, 0, BUFFER_SIZE);

            msgln("9. Client reads from InputStream and then writes it to file.");
            while (nbytes > 0) {
                file.write(buf, 0, nbytes);
                file.flush();
                nbytes = in.read(buf, 0, BUFFER_SIZE);
            }

            in.close();
            file.close();
            xfer.close();

            msgln("11. Client finished writing to file");
            msgln("******* File Successfully Retrieved using ACTIVE mode *******");

            /*
             * End TODO
             */
        } catch (IOException e) {
            msg("Exception: " + e);
            e.printStackTrace();
        }
    }
}

public void get(String[] inputs) {
    if (inputs.length == 2) {
        try {
            if (mode == Mode.PASSIVE) {

                msgln("******* Passive Mode *******");
                msgln("2. Send server input file name that has been requested");
                svr.get(inputs[1]);

                FileOutputStream f = new FileOutputStream(inputs[1]);

                Socket xfer = new Socket(serverAddress, serverSocket.getPort());
                msgln("7. Client makes the connection made with server");

                /*
                 * TODO: connect to server socket to transfer file.
                 */
                msgln("8. Client waiting for data from the server");
                InputStream in = xfer.getInputStream();
                byte[] buf = new byte[BUFFER_SIZE];
                int nbytes = in.read(buf, 0, BUFFER_SIZE);

                msgln("10. Client reads from InputStream and then writes it to file.");
                while (nbytes > 0) {
                    f.write(buf, 0, nbytes);
                    f.flush();
                    nbytes = in.read(buf, 0, BUFFER_SIZE);
                }

                in.close();
                f.close();
                xfer.close();

                msgln("12. Client finished writing to file");
                msgln("******* File Successfully Retrieved using PASSIVE mode *******");

                /*
                 * End TODO
                 */
            }
            else if (mode == Mode.ACTIVE) {
                msgln("******* ACTIVE Mode *******");
                msgln("3. Client creates an outstream to write to file with the specified filename");
                FileOutputStream f = new FileOutputStream(inputs[1]);
                msgln("4. New thread has been created");
                new Thread(new GetThread(dataChan, f)).start();
                msgln("I made it here");
                svr.get(inputs[1]);
            }
            else {
                msgln("GET: No mode set--use port or pasv command.");
            }
        } catch (Exception e) {
            msgln("Error: Inside the get function.");
            err(e);
        }
    }
}
```

File: Server.java

```
private static class GetThread implements Runnable {
    private ServerSocket dataChan = null;
    private FileInputStream file = null;

    public GetThread(ServerSocket s, FileInputStream f) {
        dataChan = s;
        file = f;
    }

    public void run() {
        /*
         * TODO: Process a client request to transfer a file.
         */
        try {
            msgln("6. Server waiting for connection from the client");
            Socket xfer = dataChan.accept();

            OutputStream out = xfer.getOutputStream();
            byte[] buf = new byte[BUFFER_SIZE];
            int nbytes = file.read(buf, 0, BUFFER_SIZE);

            msgln("9. Server writing data to send to client");
            while (nbytes > 0) {
                out.write(buf, 0, nbytes);
                out.flush();
                nbytes = file.read(buf, 0, BUFFER_SIZE);
            }

            out.close();
            file.close();
            xfer.close();

            msgln("11. Server finished sending data to client");

        } catch (IOException e) {
            msg("Exception: " + e);
            e.printStackTrace();
        }
    }
}

public void get(String file) throws IOException, FileNotFoundException, RemoteException {

    if (!valid(file)) {
        throw new IOException("Bad file name: " + file);
    } else if (mode == Mode.ACTIVE) {

        Socket xfer = new Socket(clientSocket.getAddress(), clientSocket.getPort());
        msgln("6. Server makes the connection made with client");
        /*
         * TODO: connect to client socket to transfer file.
         */
        File f = new File(path()+file);
        FileInputStream in = new FileInputStream(f);
        OutputStream out = xfer.getOutputStream();

        byte[] buf = new byte[BUFFER_SIZE];
        int nbytes = in.read(buf, 0, BUFFER_SIZE);

        msgln("8. Server writing data to send to client");
        while (nbytes > 0) {
            out.write(buf, 0, nbytes);
            out.flush();
            nbytes = in.read(buf, 0, BUFFER_SIZE);
        }

        out.close();
        in.close();
        xfer.close();

        msgln("10. Server finished sending data to client");

        /*
         * End TODO.
         */
    } else if (mode == Mode.PASSIVE) {
        msgln("3. The Path to file: " + path() + " Filename: " + file);
        msgln("4. Server makes connection with file in file system using FileInputStream");
        FileInputStream f = new FileInputStream(path() + file);
        msgln("5. New Thread has been created");
        new Thread(new GetThread(dataChan, f)).start();
    } else {
        msgln("GET: No mode set--use port or pasv command.");
    }
}
```

The code above shows how the client gets data from the server. The *get()* function above in *Client.java* takes a filename as argument and fetches that file from the server root folder in either PASSIVE mode or ACTIVE mode. The client will tell the server to do ACTIVE or PASSIVE file transfer.

In PASSIVE mode, client sends the server the filename and also creates a file with that name on its file system. The server finds that file in its file system. Then the server creates a socket and responds with the port number that the client will need connect to get the data. Server waits for the client to connect on that port. Once the client connects, it then waits for data on that port from the server that it will write to a file that it created earlier. The server will start sending data on that port, and the client will start reading data on that port and start writing to the file. After the server is finished sending data, the client will also stop reading on that port for data. The file has now successfully been retrieved from the server to the client in PASSIVE mode.

In ACTIVE mode, client acts like the server. Client will create the file on its file system. Client will also create the socket and respond with the port number that the server will connect to transfer the file to the client. Client will wait for the server to connect to the port. Once the server connects on the port, the client will wait for the server to start sending data. After the server starts sending data, the client will start reading on the port and write the data to a file. When the server is finished, the client will also finish writing to the file. The file has now successfully been retrieved from the server to the client in ACTIVE mode.

3. Put the data from the client to the server in PASSIVE and ACTIVE modes

File: Client.java

```
private static class PutThread implements Runnable {

    private ServerSocket dataChan = null;
    private FileInputStream file = null;

    public PutThread(ServerSocket s, FileInputStream f) {
        dataChan = s;
        file = f;
    }

    public void run() {
        try {
            /*
             * TODO: Complete this thread.
             */
            msgln("5. Client waiting for connection from the server");
            Socket xfer = dataChan.accept();
            OutputStream out = xfer.getOutputStream();
            byte[] buf = new byte[BUFFER_SIZE];
            int nbytes = file.read(buf, 0, BUFFER_SIZE);

            msgln("8. Client writing data to send to server");
            while (nbytes > 0) {
                out.write(buf, 0, nbytes);
                out.flush();
                nbytes = file.read(buf, 0, BUFFER_SIZE);
            }

            out.close();
            file.close();
            xfer.close();

            msgln("10. Client finished sending data to server");

        } catch (IOException e) {
            msg("Exception: " + e);
            e.printStackTrace();
        }

    }
}

public void put(String[] inputs) {
    if (inputs.length == 2) {
        try {
            /*
             * TODO: Finish put
             */
            if (mode == Mode.PASSIVE) {
                msgln("******* Passive Mode *******");
                Socket xfer = new Socket(serverAddress, serverSocket.getPort());
                msgln("2. Client creates a socket to send data to server");

                msgln("3. Client sends the file name to server");
                svr.put(inputs[1]);

                File f = new File(inputs[1]);
                FileInputStream in = new FileInputStream(f);

                OutputStream out = xfer.getOutputStream();
                byte[] buf = new byte[BUFFER_SIZE];
                int nbytes = in.read(buf, 0, BUFFER_SIZE);

                msgln("8. Client writing data to send to server");
                while (nbytes > 0) {
                    out.write(buf, 0, nbytes);
                    out.flush();
                    nbytes = in.read(buf, 0, BUFFER_SIZE);
                }

                out.close();
                in.close();
                xfer.close();

                msgln("10. Finished sending data to server on port " + serverSocket.getPort());

                msgln("******* File Successfully placed on server using PASSIVE mode *******");


            } else if (mode == Mode.ACTIVE) {

                msgln("******* ACTIVE Mode *******");
                msgln("3. Client connects to file with the specified filename: " + inputs[1]);
                FileInputStream in = new FileInputStream(inputs[1]);
                msgln("4. New thread has been created.");
                new Thread(new PutThread(dataChan, in)).start();
                svr.put(inputs[1]);

                msgln("******* File Successfully placed on server using ACTIVE mode *******");

            } else {
                msgln("PUT: No mode set--use port or pasv command.");
            }

        } catch (Exception e) {
            err(e);
        }
    }
}
```

File: Server.java

```
private static class PutThread implements Runnable {

    private ServerSocket dataChan = null;
    private FileOutputStream file = null;

    public PutThread(ServerSocket s, FileOutputStream f) {
        dataChan = s;
        file = f;
    }

    public void run() {
        try {
            msgln("6. Server listens on socket");
            Socket xfer = dataChan.accept();

            msgln("7. Server waiting for data from the client");
            InputStream in = xfer.getInputStream();

            byte[] buf = new byte[BUFFER_SIZE];
            int nbytes = in.read(buf, 0, BUFFER_SIZE);

            msgln("9. Server receives data from client and then writes it to file.");
            while (nbytes > 0) {
                file.write(buf, 0, nbytes);
                file.flush();
                nbytes = in.read(buf, 0, BUFFER_SIZE);
            }

            in.close();
            file.close();
            xfer.close();

            msgln("11. Server finished writing to file");

            msgln("******* File Successfully placed on server using ACTIVE mode *******");

        } catch (IOException e) {
            msg("Exception: " + e);
            e.printStackTrace();
        }
    }
}


public void put(String file) throws IOException, FileNotFoundException, RemoteException {
    /*
     * TODO: Finish put
     */
    if (!valid(file)) {
        throw new IOException("Bad file name: " + file);
    } else if (mode == Mode.ACTIVE) {

        Socket xfer = new Socket(clientSocket.getAddress(), clientSocket.getPort());
        msgln("6. Server creates the connection with client");

        File f = new File(path()+file);
    FileOutputStream out = new FileOutputStream(f);

    msgln("7. Server waiting for data from the client");
        InputStream in = xfer.getInputStream();

    byte[] buf = new byte[BUFFER_SIZE];
        int nbytes = in.read(buf, 0, BUFFER_SIZE);

        msgln("9. Server reads from InputStream and then writes it to file");
        while (nbytes > 0) {
            out.write(buf, 0, nbytes);
            out.flush();
            nbytes = in.read(buf, 0, BUFFER_SIZE);
        }
        out.close();
        in.close();
        xfer.close();
        msgln("11. Server Finished writing to the OutputStream file");
        msgln("******* File Successfully placed on server using PASSIVE mode *******");
    } else if (mode == Mode.PASSIVE) {
        msgln("4. Server create FileOutputStream in Path: " + path() + " File: " + file);
        FileOutputStream f = new FileOutputStream(path() + file);
        msgln("5. New thread has been created");
        new Thread(new PutThread(dataChan, f)).start();
    }
    else {
        msgln("PUT: No mode set--use port or pasv command.");
    }
}
```

The code above shows how the client puts data on the server in ACTIVE and PASSIVE modes. The *put()* function in *Client.java* will take the filename to put on server as argument and place that file in the root folder on the server.

In PASSIVE mode, the server will create the socket and open port where it will receive data. Next the client sends the filename to the server. The server creates that file on the server file system. Now the server is connecting to that port and waiting for data from the client side. Client will read data from the file it wants to transfer and start sending it to the server. The server will read that data and write it to the file. Once the client the finished sending the data, the server will be finished writing the data to the file. The file has now successfully been placed on the server from client in PASSIVE mode.

In ACTIVE mode, the clients connect with the file that will be transferred to the server. Client will create the socket and the port that will be used to transfer the data. Next the client is waiting for connection to be made from the server. Server will connect to the socket and now wait for data to be received from the client. The client will start sending data and the server will start reading that data on the port and write to the file. Once the client the finished sending the data, the server will be finished writing the data to the file. The file has now successfully been placed on the server from client in ACTIVE mode.

4. Other code added

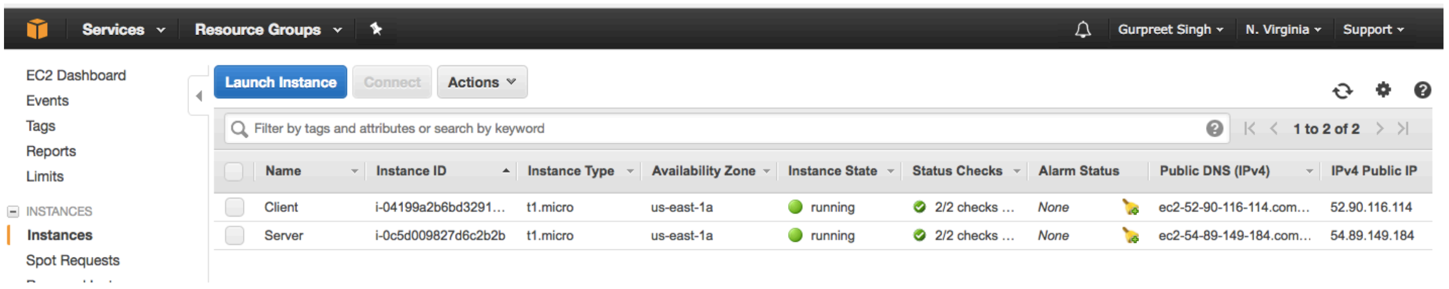
File: Client.java

```
private static InetAddress host;
final static int backlog = 5;
...

String clientIp = "172.31.39.28";
Client.host = InetAddress.getByName(clientIp);
...

dataChan = new ServerSocket(0, backlog, Client.host);
```

I had to add the code above because when I moved the code to EC2 instances, I was unable to connect to the client. Client was providing its address as 0.0.0.0 to the server, which is wrong. You will have to modify the clientIp if you will test this on EC2 machine. I commented this code out for local versions testing.

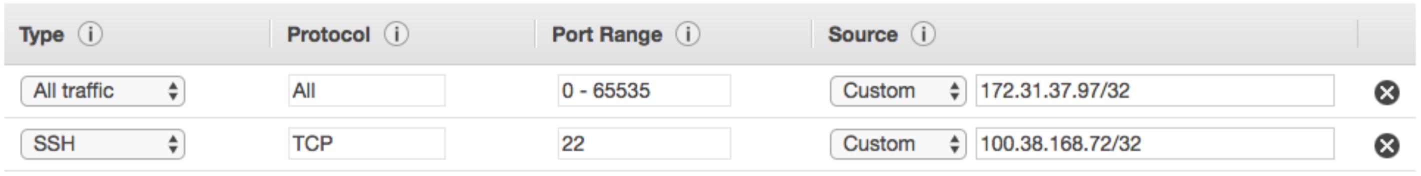
5. Testing

The code above has been tested in AWS EC2 instances. I created two EC2 instance, client and server. The jar files generated by the program were copied to these instances. Next the policy and shell files were created from the jar files and the server and client program were started.



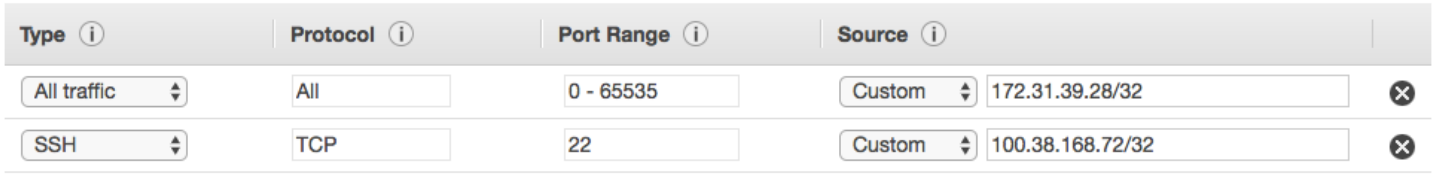
	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	Client	i-04199a2b6bd3291…	t1.micro	us-east-1a	● running	✓ 2/2 checks …	None	ec2-52-90-116-114.com…	52.90.116.114
	Server	i-0c5d009827d6c2b2b	t1.micro	us-east-1a	● running	✓ 2/2 checks …	None	ec2-54-89-149-184.com…	54.89.149.184

Figure 1. AWS EC2 Instances



Type	Protocol	Port Range	Source	
All traffic	All	0 - 65535	Custom	172.31.37.97/32
SSH	TCP	22	Custom	100.38.168.72/32

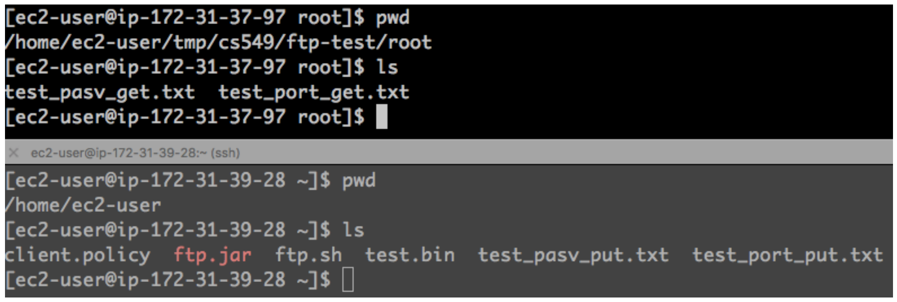
Figure 2. Client Security Group



Type	Protocol	Port Range	Source	
All traffic	All	0 - 65535	Custom	172.31.39.28/32
SSH	TCP	22	Custom	100.38.168.72/32

Figure 3. Server Security Group

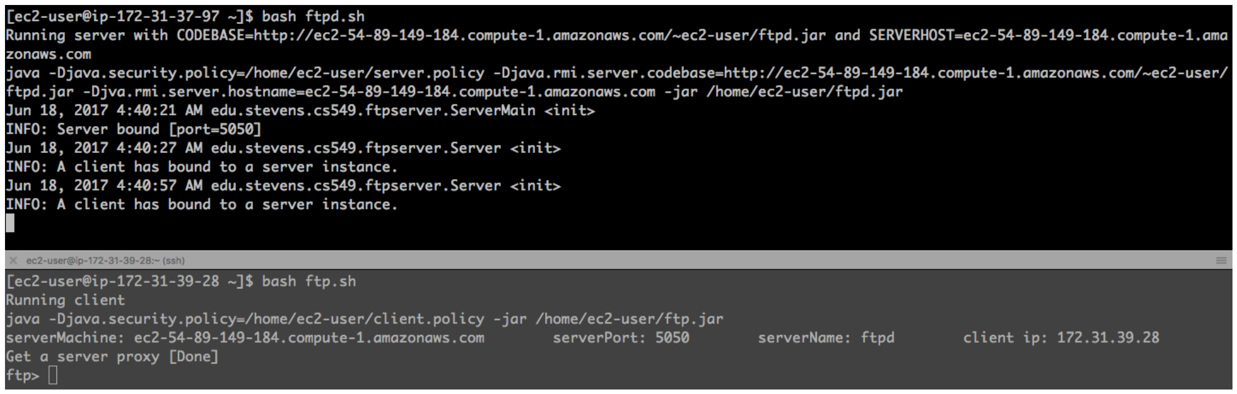
Figure 2 and 3 show the security groups for each of the instances. Both instances allow SSH traffic from my personal computer. Server instance allow inbound traffic on all ports from the client instance for PASSIVE file transfer. Client instance allow inbound traffic on all ports from the server instance for ACTIVE file transfer.



```
[ec2-user@ip-172-31-37-97 root]$ pwd
/home/ec2-user/tmp/cs549/ftp-test/root
[ec2-user@ip-172-31-37-97 root]$ ls
test_pasv_get.txt  test_port_get.txt
[ec2-user@ip-172-31-37-97 root]$
```

```
ec2-user@ip-172-31-39-28:~ (ssh)
[ec2-user@ip-172-31-39-28 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-39-28 ~]$ ls
client.policy  ftp.jar  ftp.sh  test.bin  test_pasv_put.txt  test_port_put.txt
[ec2-user@ip-172-31-39-28 ~]$
```

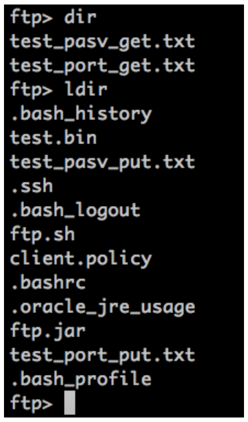
Figure 4. File system for the client and server instance



```
[ec2-user@ip-172-31-37-97 ~]$ bash ftpd.sh
Running server with CODEBASE=http://ec2-54-89-149-184.compute-1.amazonaws.com/~ec2-user/ftpd.jar and SERVERHOST=ec2-54-89-149-184.compute-1.ama
zonaws.com
java -Djava.security.policy=/home/ec2-user/server.policy -Djava.rmi.server.codebase=http://ec2-54-89-149-184.compute-1.amazonaws.com/~ec2-user/
ftpd.jar -Djva.rmi.server.hostname=ec2-54-89-149-184.compute-1.amazonaws.com -jar /home/ec2-user/ftpd.jar
Jun 18, 2017 4:40:21 AM edu.stevens.cs549.ftpserver.ServerMain <init>
INFO: Server bound [port=5050]
Jun 18, 2017 4:40:27 AM edu.stevens.cs549.ftpserver.Server <init>
INFO: A client has bound to a server instance.
Jun 18, 2017 4:40:57 AM edu.stevens.cs549.ftpserver.Server <init>
INFO: A client has bound to a server instance.
```

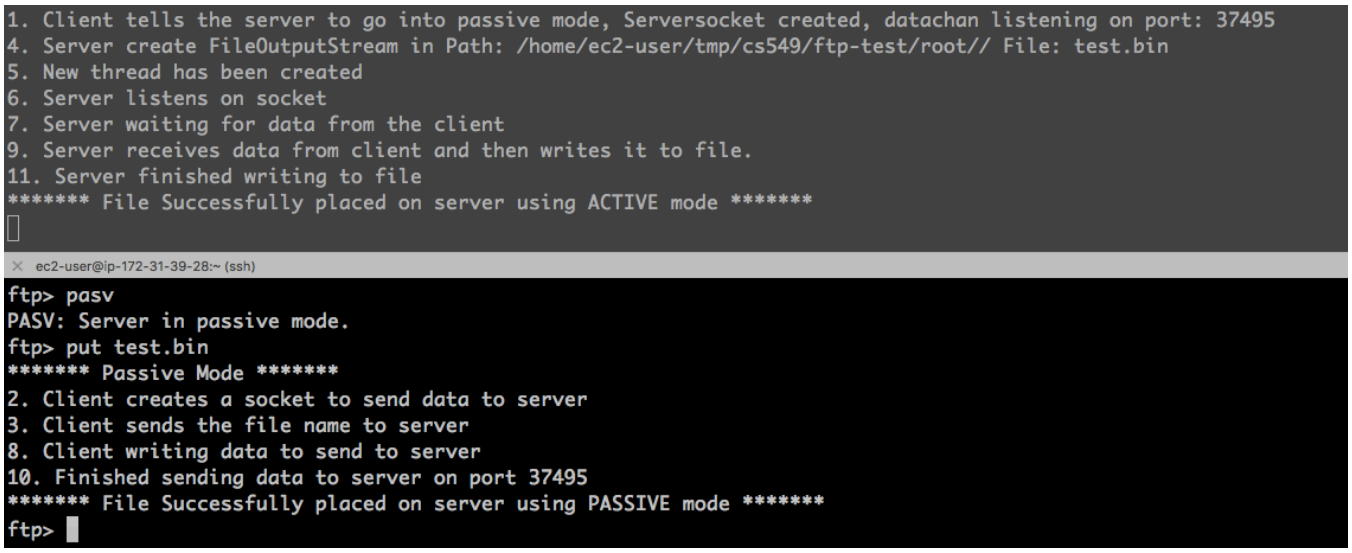
```
ec2-user@ip-172-31-39-28:~ (ssh)
[ec2-user@ip-172-31-39-28 ~]$ bash ftp.sh
Running client
java -Djava.security.policy=/home/ec2-user/client.policy -jar /home/ec2-user/ftp.jar
serverMachine: ec2-54-89-149-184.compute-1.amazonaws.com        serverPort: 5050        serverName: ftpd        client ip: 172.31.39.28
Get a server proxy [Done]
ftp>
```

Figure 5. Client and Server EC2 instances



```
ftp> dir
test_pasv_get.txt
test_port_get.txt
ftp> ldir
.bash_history
test.bin
test_pasv_put.txt
.ssh
.bash_logout
ftp.sh
client.policy
.bashrc
.oracle_jre_usage
ftp.jar
test_port_put.txt
.bash_profile
ftp>
```

Figure 6. Client reads the server file system and its own file system

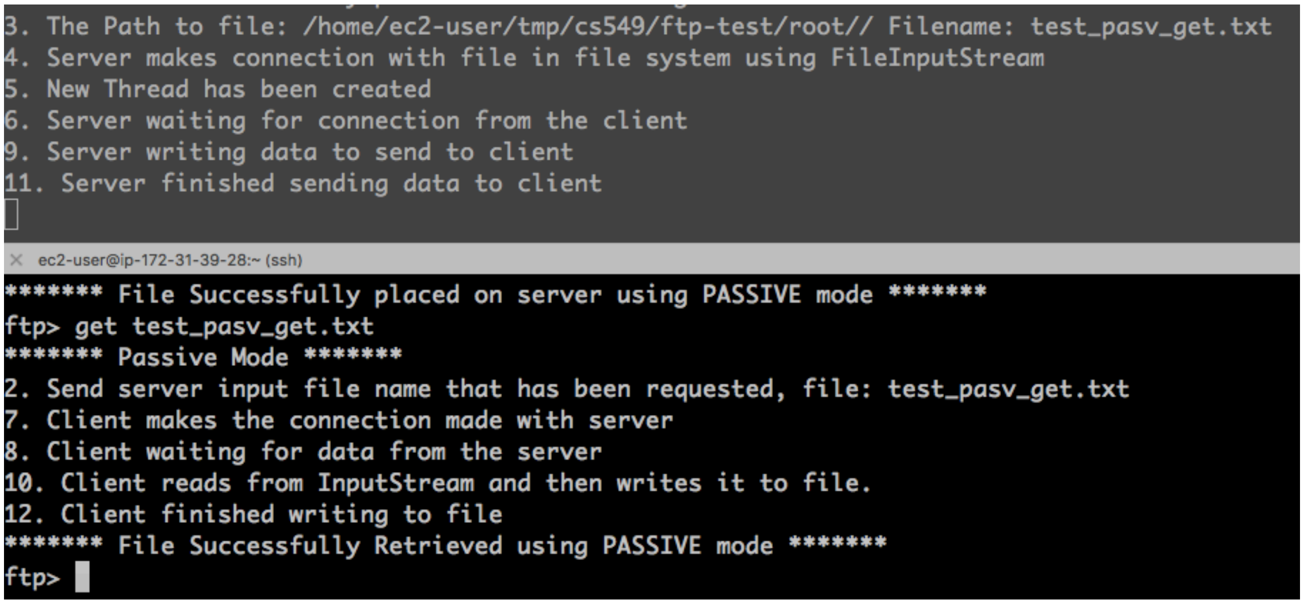


```
1. Client tells the server to go into passive mode, Serversocket created, datachan listening on port: 37495
4. Server create FileOutputStream in Path: /home/ec2-user/tmp/cs549/ftp-test/root// File: test.bin
5. New thread has been created
6. Server listens on socket
7. Server waiting for data from the client
9. Server receives data from client and then writes it to file.
11. Server finished writing to file
******* File Successfully placed on server using ACTIVE mode *******
```

```
ec2-user@ip-172-31-39-28:~ (ssh)
```

```
ftp> pasv
PASV: Server in passive mode.
ftp> put test.bin
******* Passive Mode *******
2. Client creates a socket to send data to server
3. Client sends the file name to server
8. Client writing data to send to server
10. Finished sending data to server on port 37495
******* File Successfully placed on server using PASSIVE mode *******
ftp>
```

Figure 7. Passive file transfer from the client to server (PUT test)

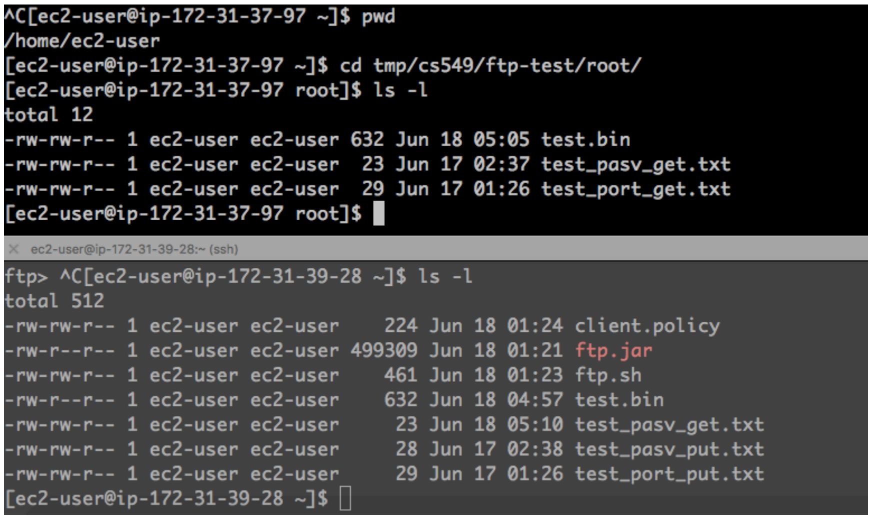


```
3. The Path to file: /home/ec2-user/tmp/cs549/ftp-test/root// Filename: test_pasv_get.txt
4. Server makes connection with file in file system using FileInputStream
5. New Thread has been created
6. Server waiting for connection from the client
9. Server writing data to send to client
11. Server finished sending data to client
```

```
ec2-user@ip-172-31-39-28:~ (ssh)
```

```
******* File Successfully placed on server using PASSIVE mode *******
ftp> get test_pasv_get.txt
******* Passive Mode *******
2. Send server input file name that has been requested, file: test_pasv_get.txt
7. Client makes the connection made with server
8. Client waiting for data from the server
10. Client reads from InputStream and then writes it to file.
12. Client finished writing to file
******* File Successfully Retrieved using PASSIVE mode *******
ftp>
```

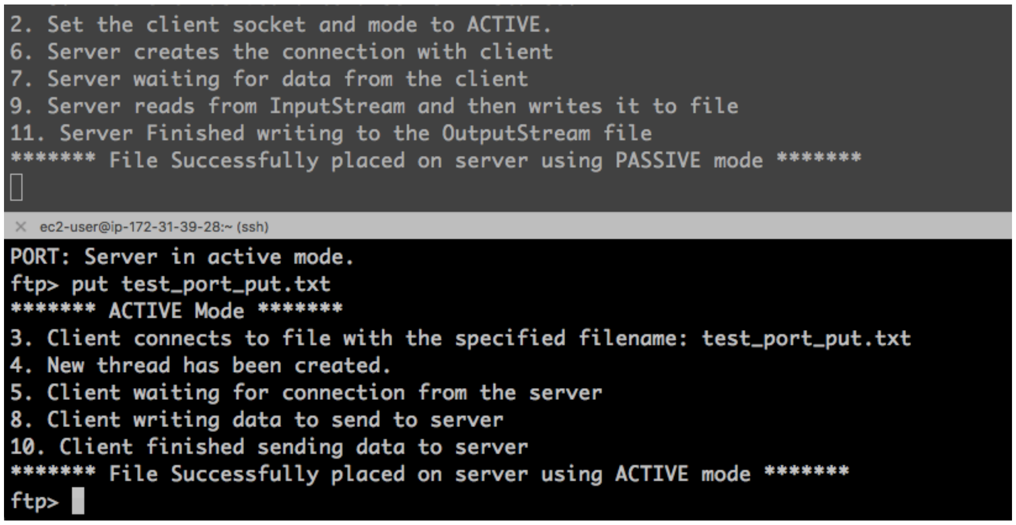
Figure 8. Passive file transfer from the server to the client (GET test)


```
^C[ec2-user@ip-172-31-37-97 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-37-97 ~]$ cd tmp/cs549/ftp-test/root/
[ec2-user@ip-172-31-37-97 root]$ ls -l
total 12
-rw-rw-r-- 1 ec2-user ec2-user 632 Jun 18 05:05 test.bin
-rw-rw-r-- 1 ec2-user ec2-user  23 Jun 17 02:37 test_pasv_get.txt
-rw-rw-r-- 1 ec2-user ec2-user  29 Jun 17 01:26 test_port_get.txt
[ec2-user@ip-172-31-37-97 root]$
```

```
ftp> ^C[ec2-user@ip-172-31-39-28 ~]$ ls -l
total 512
-rw-rw-r-- 1 ec2-user ec2-user    224 Jun 18 01:24 client.policy
-rw-r--r-- 1 ec2-user ec2-user 499309 Jun 18 01:21 ftp.jar
-rw-rw-r-- 1 ec2-user ec2-user    461 Jun 18 01:23 ftp.sh
-rw-r--r-- 1 ec2-user ec2-user    632 Jun 18 04:57 test.bin
-rw-rw-r-- 1 ec2-user ec2-user     23 Jun 18 05:10 test_pasv_get.txt
-rw-rw-r-- 1 ec2-user ec2-user     28 Jun 17 02:38 test_pasv_put.txt
-rw-rw-r-- 1 ec2-user ec2-user     29 Jun 17 01:26 test_port_put.txt
[ec2-user@ip-172-31-39-28 ~]$
```

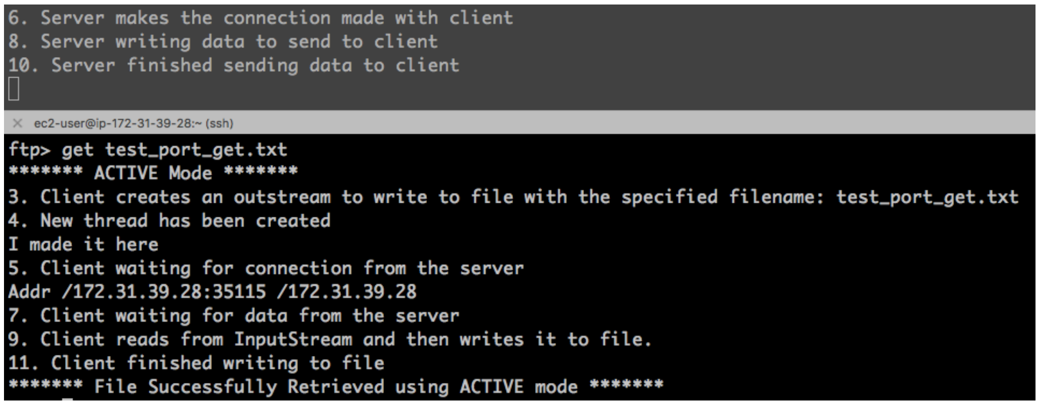
Figure 9. File listed on client and server instances after Passive transfer (compare with Figure 4)

Figure 9 shows that *test.bin* has been successfully moved from client to server in passive mode and *test_pasv_get.txt* has been successfully moved from the server to the client in passive mode.


```
2. Set the client socket and mode to ACTIVE.
6. Server creates the connection with client
7. Server waiting for data from the client
9. Server reads from InputStream and then writes it to file
11. Server Finished writing to the OutputStream file
******* File Successfully placed on server using PASSIVE mode *******
```

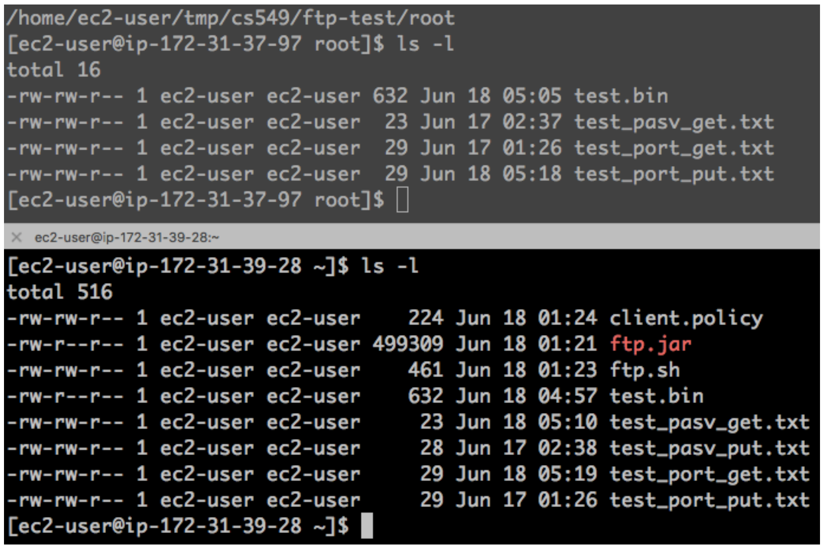
```
PORT: Server in active mode.
ftp> put test_port_put.txt
******* ACTIVE Mode *******
3. Client connects to file with the specified filename: test_port_put.txt
4. New thread has been created.
5. Client waiting for connection from the server
8. Client writing data to send to server
10. Client finished sending data to server
******* File Successfully placed on server using ACTIVE mode *******
ftp>
```

Figure 10. Active File transfer from the client to server (PUT)


```
6. Server makes the connection made with client
8. Server writing data to send to client
10. Server finished sending data to client
```

```
ftp> get test_port_get.txt
******* ACTIVE Mode *******
3. Client creates an outstream to write to file with the specified filename: test_port_get.txt
4. New thread has been created
I made it here
5. Client waiting for connection from the server
Addr /172.31.39.28:35115 /172.31.39.28
7. Client waiting for data from the server
9. Client reads from InputStream and then writes it to file.
11. Client finished writing to file
******* File Successfully Retrieved using ACTIVE mode *******
```

Figure 11. Active file transfer from the server to client (GET)



```
/home/ec2-user/tmp/cs549/ftp-test/root
[ec2-user@ip-172-31-37-97 root]$ ls -l
total 16
-rw-rw-r-- 1 ec2-user ec2-user 632 Jun 18 05:05 test.bin
-rw-rw-r-- 1 ec2-user ec2-user  23 Jun 17 02:37 test_pasv_get.txt
-rw-rw-r-- 1 ec2-user ec2-user  29 Jun 17 01:26 test_port_get.txt
-rw-rw-r-- 1 ec2-user ec2-user  29 Jun 18 05:18 test_port_put.txt
[ec2-user@ip-172-31-37-97 root]$
```

ec2-user@ip-172-31-39-28:~

```
[ec2-user@ip-172-31-39-28 ~]$ ls -l
total 516
-rw-rw-r-- 1 ec2-user ec2-user    224 Jun 18 01:24 client.policy
-rw-r--r-- 1 ec2-user ec2-user 499309 Jun 18 01:21 ftp.jar
-rw-rw-r-- 1 ec2-user ec2-user    461 Jun 18 01:23 ftp.sh
-rw-r--r-- 1 ec2-user ec2-user    632 Jun 18 04:57 test.bin
-rw-rw-r-- 1 ec2-user ec2-user     23 Jun 18 05:10 test_pasv_get.txt
-rw-rw-r-- 1 ec2-user ec2-user     28 Jun 17 02:38 test_pasv_put.txt
-rw-rw-r-- 1 ec2-user ec2-user     29 Jun 18 05:19 test_port_get.txt
-rw-rw-r-- 1 ec2-user ec2-user     29 Jun 17 01:26 test_port_put.txt
[ec2-user@ip-172-31-39-28 ~]$
```

Figure 12. Files in client and server file system after Active file transfers

6. Additional Instructions

Refer to the video demonstration to make the code work on EC2 instances. You will need to change the local IP address of the client EC2 instance and the IP address of the server EC2 instance.

7. Conclusion

ACTIVE and PASSIVE file transfer work differently and this assignment has made that super clear. In ACTIVE mode, the client acts like the server and tells the port number to the server to connect to. Server than initiates the data connection and client accepts to complete the transaction. In PASSIVE mode, server creates the socket and responds with the port number to the client. Client initiates the data connection and server accepts to complete the transaction.