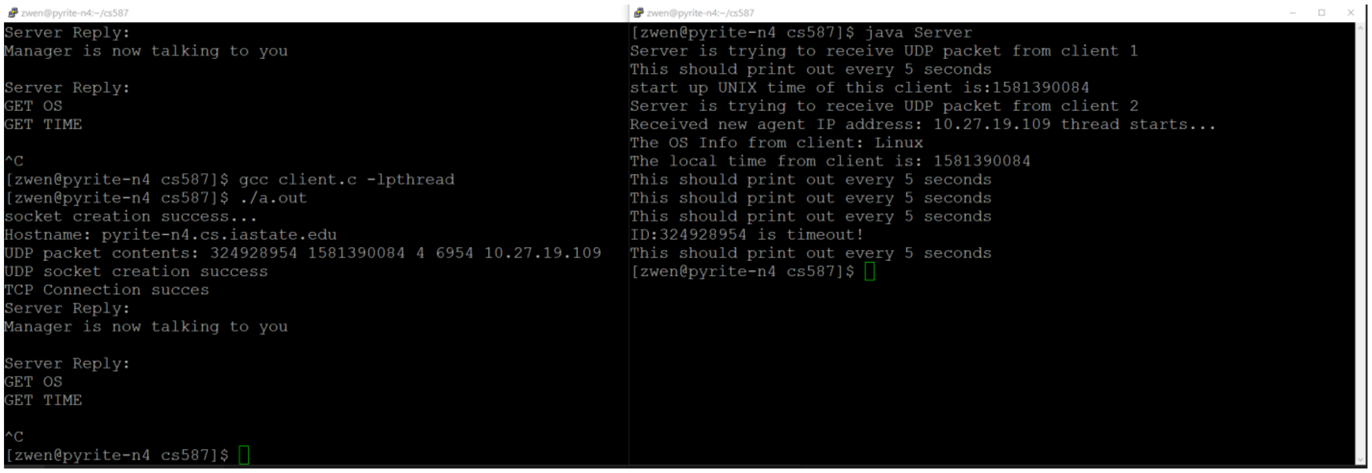
Screenshots and demonstration:

Overall 1-1 time-out Demo:

The following screenshots demonstrate my program meet most of requirements directly for this assignment, the other requirement will be explained by pseudo code and other instruction section:

- Agent is able to generate ID, record start up time, set time interval, create port number, and get host IP address. ID is randomly generated during startup. Port number is also randomly generated since we are testing in local machine, so port number cannot be the same if we need to start multiple agents.
- All BEACON struct information is packed into one UDP packet successfully.
- UDP connection is successful.
- TCP connection is successful.
- Manager is able to continuously wait and receive UDP packets from agents (“clients”).
- Manager is able to display UDP packet contents from agent: here we use IP address and startup time as example.
- Manager is able to call getOS and getTime function and get data from agent by TCP channel which is “Linux” and UNIX time “1581390084” in this case.
- Manager is able to check stored agent’s health status periodically every 5 seconds in another thread (for easy demonstration I picked small time interval and check period). After pause agent side, after a while, manager will display that agent ID and say it is timed out.



```
Server Reply:
Manager is now talking to you

Server Reply:
GET OS
GET TIME

^C
[zwen@pyrite-n4 cs587]$ gcc client.c -lpthread
[zwen@pyrite-n4 cs587]$ ./a.out
socket creation success...
Hostname: pyrite-n4.cs.iastate.edu
UDP packet contents: 324928954 1581390084 4 6954 10.27.19.109
UDP socket creation success
TCP Connection succes
Server Reply:
Manager is now talking to you

Server Reply:
GET OS
GET TIME

^C
[zwen@pyrite-n4 cs587]$
```

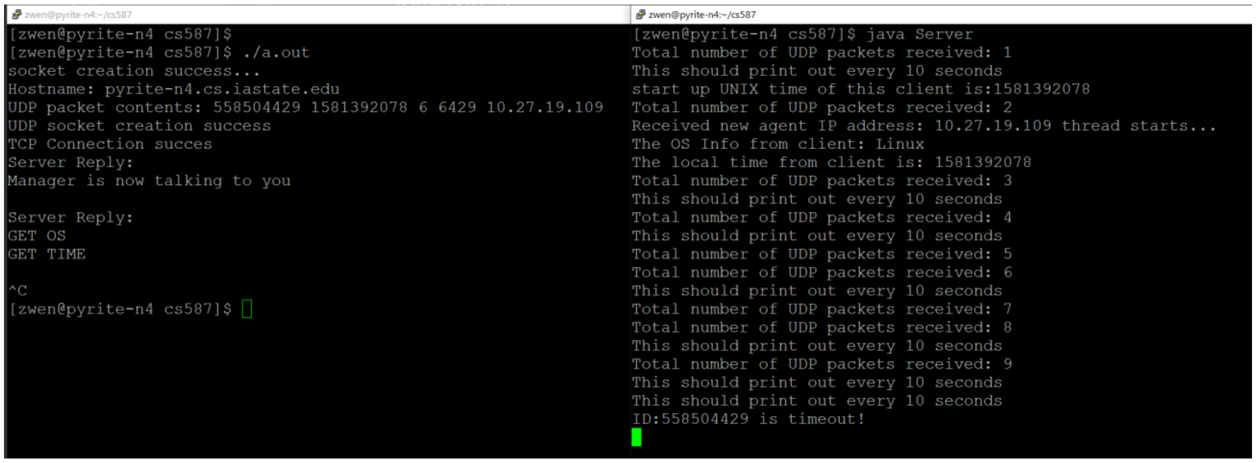
```
[zwen@pyrite-n4 cs587]$ java Server
Server is trying to receive UDP packet from client 1
This should print out every 5 seconds
start up UNIX time of this client is:1581390084
Server is trying to receive UDP packet from client 2
Received new agent IP address: 10.27.19.109 thread starts...
The OS Info from client: Linux
The local time from client is: 1581390084
This should print out every 5 seconds
This should print out every 5 seconds
This should print out every 5 seconds
ID:324928954 is timeout!
This should print out every 5 seconds
[zwen@pyrite-n4 cs587]$
```

UDP no time-out DEMO:

The next screenshot demonstrates:

- As you can see the time interval is set to 6 seconds. So every 6 seconds manager display that it get a new UDP packet from that client and update the count.
- Only the first time that manager get UDP packet from this client display manager get new agent by showing its startup time and IP address. Therefore, the correctness of check ID and add to the clients table is proved.

- I can display ID or whatever info agent is provided, based on time concern, I think start up time and IP information are strong enough to prove agents' identity (could also be replaced by ID).



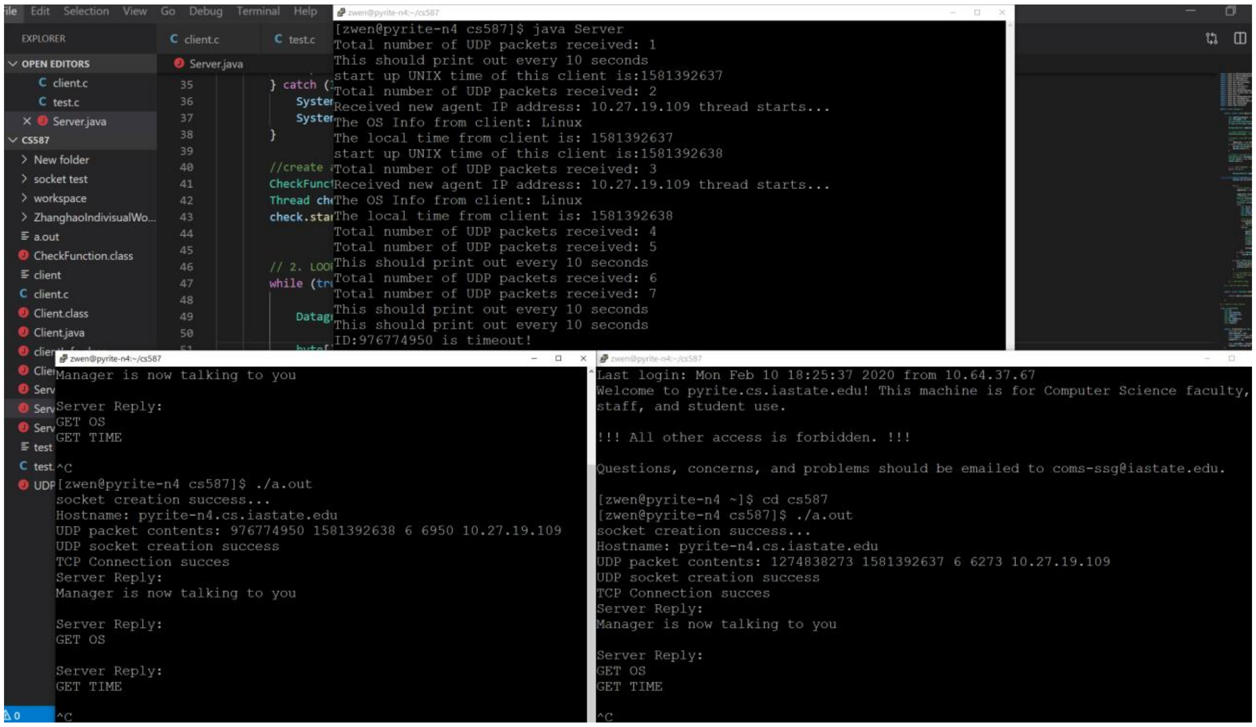
```
[zwen@pyrite-n4 cs587]$
[zwen@pyrite-n4 cs587]$ ./a.out
socket creation success...
Hostname: pyrite-n4.cs.iastate.edu
UDP packet contents: 558504429 1581392078 6 6429 10.27.19.109
UDP socket creation success
TCP Connection succes
Server Reply:
Manager is now talking to you

Server Reply:
GET OS
GET TIME

^C
[zwen@pyrite-n4 cs587]$
```

```
[zwen@pyrite-n4 cs587]$ java Server
Total number of UDP packets received: 1
This should print out every 10 seconds
start up UNIX time of this client is:1581392078
Total number of UDP packets received: 2
Received new agent IP address: 10.27.19.109 thread starts...
The OS Info from client: Linux
The local time from client is: 1581392078
Total number of UDP packets received: 3
This should print out every 10 seconds
Total number of UDP packets received: 4
This should print out every 10 seconds
Total number of UDP packets received: 5
Total number of UDP packets received: 6
This should print out every 10 seconds
Total number of UDP packets received: 7
Total number of UDP packets received: 8
This should print out every 10 seconds
Total number of UDP packets received: 9
This should print out every 10 seconds
This should print out every 10 seconds
ID:558504429 is timeout!
```

Multiple-agents DEMO:



```
[zwen@pyrite-n4 cs587]$ java Server
Total number of UDP packets received: 1
This should print out every 10 seconds
start up UNIX time of this client is:1581392637
Total number of UDP packets received: 2
Received new agent IP address: 10.27.19.109 thread starts...
The OS Info from client: Linux
The local time from client is: 1581392637
start up UNIX time of this client is:1581392638
Total number of UDP packets received: 3
Received new agent IP address: 10.27.19.109 thread starts...
The OS Info from client: Linux
The local time from client is: 1581392638
Total number of UDP packets received: 4
Total number of UDP packets received: 5
This should print out every 10 seconds
Total number of UDP packets received: 6
Total number of UDP packets received: 7
This should print out every 10 seconds
This should print out every 10 seconds
ID:976774950 is timeout!
```

```
Manager is now talking to you

Server Reply:
GET OS
GET TIME

^C
[zwen@pyrite-n4 cs587]$ ./a.out
socket creation success...
Hostname: pyrite-n4.cs.iastate.edu
UDP packet contents: 976774950 1581392638 6 6950 10.27.19.109
UDP socket creation success
TCP Connection succes
Server Reply:
Manager is now talking to you

Server Reply:
GET OS

Server Reply:
GET TIME

^C
```

```
Last login: Mon Feb 10 18:25:37 2020 from 10.64.37.67
Welcome to pyrite.cs.iastate.edu! This machine is for Computer Science faculty,
staff, and student use.

!!! All other access is forbidden. !!!

Questions, concerns, and problems should be emailed to coms-ssg@iastate.edu.

[zwen@pyrite-n4 ~]$ cd cs587
[zwen@pyrite-n4 cs587]$ ./a.out
socket creation success...
Hostname: pyrite-n4.cs.iastate.edu
UDP packet contents: 1274838273 1581392637 6 6273 10.27.19.109
UDP socket creation success
TCP Connection succes
Server Reply:
Manager is now talking to you

Server Reply:
GET OS
GET TIME

^C
```

How to run the program:

The program is developed in pyrite.cs.iastate.edu

To compile the c code: gcc client.c -lpthread

To run the c file: ./a.out

To compile the Java code: javac Server.java

To run the java code: java Server

Design details:

Overall workflow:

- ❖ Agent send UDP packet to manager
    - Build UDP socket
        - Create socket
        - Java listen port and c know the port
    - Create struct and relevant information
        - int ID;
        - int startUpTime;
        - int timeInterval;
        - int cmdPort;
        - char IP[4];
    - launch thread to send UDP packet periodically
- ❖ Manager listen UDP packet, process data, and build TCP connection
    - Create thread implementation, data structure to store clients information
        - ArrayList<Thread> clientThreads
        - ArrayList<ClientsHandler> clients
        - Hashtable<Integer, clientInfo> agentsTable
        - class CheckFunction implements Runnable
        - class ClientsHandler implements Runnable
        - class clientInfo
    - Process data:
        - Use 1 thread to listen UDP.
        - if UDP packet ID is new, create new client and add it to table and build TCP connetion, if not, update timeout.
        - Store all data to the table.
    - TCP channel
- ❖ Manager send command to agents through TCP
    - Send getOS, getLocalTime
    - Receive and printout

Pseudo code C

```
struct BEACON
declare function
char* getIPAddress()
void GetLocalTime
void GetLocalOS
void *BeaconSender(void *vargp){
        get myID, my_start_time, my_time_interval, bport, *IP;
        struct BEACON myBeacon = {myID, my_start_time, my_time_interval, bport, *IP};
        convert data from all kinds of type to char array
        sprintf convert from int type to char array
        srrcat link all char array to the message that going to send by UDP
        get message
    Build UDP connection;
    While(1){
        Send message
        sleep(time_interval)
    }
}
main(){
    create BeaconSender thread by pthread_t;
    build TCP connection
    listen to the port that created before cmdPort;
    while(1){
        accept message from manager
        strcmp compare string from server to local command
        if(strcmp(server_reply, "GET OS")==0){
            send OS
        }
        if(strcmp(server_reply, "GET TIME")==0){
            send TIME
        }
    }
}
```

Pseudo code Java

```
Main{
    ArrayList<ClientsHandler> clients = new ArrayList<ClientsHandler>();
    ArrayList<Thread> clientThreads = new ArrayList<Thread>();
    Hashtable<Integer, clientInfo> agentsTable = new Hashtable<Integer, clientInfo>();
    CheckFunction checkHealth = new CheckFunction(agentsTable,clientThreads);
    Thread check = new Thread(checkHealth);
    While(true){
        udpPacket = new DatagramPacket(receive, receive.length);
        udpSocket.receive(udpPacket);
        threadID=0;
        process received string to each id, start time, time, interval, port, client_ip
        if(have same id){
            updatetimeout();
        }else{
            Add all info to the table
```

```
                    Print out get new agent message
                    Create TCP connection
                    Launch new thread
                    ClientsHandler()
                    clientThreads.add(threadID,t)
                    threadID++
                }

        }

}

Updatetimeout{ (thread)
        timeout = (int)(new Timestamp(System.currentTimeMillis()).getTime()/ 1000L) + 2*timeinterval;
}
class ClientsHandler implements Runnable{
                ClientsHandler (this.port = port, this.ip = ip)
                run(){
                    try{
                        clientSocket = new Socket("localhost",port);
                        out = new printWriter();
                        out.println(cmd)
                        out.flush()
                        in = scanner()
                        print(in.nextline) get OS and Time
                    }catch{IOE exception}
                }
}
class CheckFunction implements Runnable{
        pass agentstable and clientThreads
        run(){
                while(true){
                    go through all table to check if each agent is timed out
                    if(ID.timeout){
                        table.remove(ID)
                    }
                Thread.sleep(10000);
                }
        }
}
```

Other details:

- The port number is generated randomly between 6000 – 6999 and use current UNIX time as seed to make sure generate different random sequence when start up a new agent.
- ID is also use current UNIX time as seed to make sure generate different random sequence when start up a new agent.
- Manager is able to check ID, if ID does not exist, then create new thread for that agent to build TCP connection, if exist, then update new timeout information.
- **Thread usage discussion:** In my implementation from manager side, I started with use different threads to handle each UDP packet and use thread to handle TCP connection. Although this could be "ideal" to handle situation like manager get 2 UDP packets at the same time while one thread is only able to process one packet at a time and the other UDP packet could get dropped. However, it is too tedious and maybe not that important in reality. If one packet dropped, then the agent could send again after the pre-defined time interval. So, in my later implementation I changed only use 1 thread to listen UDP. Based on the nature of TCP, there is not doubt that use different thread to handle each TCP connection. When one TCP connection terminated, then close that thread to save memory (there is limitation on thread usage). To accomplish this, I used ArrayList to handle all threads and control its start and close.
- Create a new thread in java to check health status. Use system time, time interval, and last time one UDP packet time to determine whether timeout or not. If time out, then remove from agentsTable.
- Create object clientInfo to store all data from struct. And use clientInfo as data type in the Hashtable. Hashtable<Integer, clientInfo> first integer act as a key and overall to make data communication between different functions easier.