Computer Science 391 – Fall 2013
Assignment #3

(80 points) **Due: Sunday, 11/24/13 at 11:59PM in D2L**

You may work in pairs on this assignment, if you so choose. Clearly state who your partner is in your README file, and make only one submission for both of you. You will both receive the same grade.

The goal of this programming assignment is to develop a simple P2P file sharing application.

You should develop a program called `peer` that acts as a file sharing peer in a P2P network. Every `peer` has a designated directory where text files meant for sharing are stored. Once a peer has joined a network, it can download a specified file from another `peer` having that file. To simplify the problem, we assume that all file names and peer names are implicitly unique across the network.

Here are some more details on the `peer`:

1. The `peer` is started by providing its peer-name, peer-IP-address, peer-port, path to designated directory, and the IP address and port number of another peer who is already in the network. This data is provided as command line arguments (6 arguments, in this order). If the last two arguments are omitted (4 arguments in total), then this peer is starting as the *first node* in a *new* P2P network.
2. Every `peer`, after joining a network, will also continuously listen on a UDP port (called the *lookup* port) and a TCP port (called the *file transfer* port). The lookup port runs on the peer-port specified when the peer was started. The file transfer port is chosen privately by the peer. (Your code **must** choose this port as `lookup port + 1`)
3. Assume that peer A joins a P2P network by being (externally) provided the details of an existing peer B. Peer A then makes a UDP connection to the lookup port of B asking to join the network. B unconditionally accepts the request of A, and adds A as its neighbor; B then sends an acknowledgment to A *on the ephemeral port that A sent the request from*. A has now joined the network with B as its neighbor.
4. After joining the network, the `peer` will read the files in its designated directory and create a list of the files it has. It also starts listening on the lookup and file transfer ports.
5. Then the `peer` should also continually respond to the following menu commands:
    a. `status` – provides a list of the details of its neighbors, and a list of the files it currently has.
    b. `find <filename>` – attempts to discover the location of this file on the network. If the peer has the file locally, report this; otherwise do a network-wide lookup based on *controlled flooding* as follows:
        i. A lookup message for this file is flooded to all neighbors of the `peer` by sending a UDP datagram message to the lookup port of each neighbor. The lookup message is keyed by the source `peer` name and a sequence number (maintained by the source).

ii. Every neighbor/peer receiving this datagram for the first time will first do a local lookup to see if it has this file;
    1. if it does, it sends a UDP datagram message to the lookup port of the source `peer`, informing the source that the file is available *here* (peer-name, peer-ip, peer-port, file-transfer-port). Note that the file transfer port number it is listening on should be included.
    2. If it does not have the file, it will flood the message to all its neighbors (except the neighbor from which it received the packet).
iii. If a neighbor/peer has received this same message before (based on peer-name, sequence number), it simply discards the message.

Since the response to the peer's UDP lookup request will arrive at the lookup port (*and not the ephemeral port the request was sent on*), your program can either go back to the menu display right away without waiting for responses, or wait a reasonable time interval for responses to be displayed before showing the menu again. In either case, it will be up to the lookup port handling code to display any responses received. For all valid responses, you should print out *all the details* of the peer who has this file.

c. `get <filename> <target-peer-ip> <target-file-transfer-port>` – attempts to download this file from the specific target peer. The peer will open a TCP connection to the file-transfer port of the target peer and request the file. The target peer will then send the file, if it has it, or respond with an error message. Once a file is obtained, it is added to the list of files now available on that peer.

d. `quit` – terminates the `peer`, by notifying its neighbors to remove it from their neighbor list through a UDP message to the lookup port of all its neighbors. Don't worry about receiving any acknowledgments to this message. Do not also worry about other peers becoming disconnected from the network as a side effect, or the network fragmenting.

6. Note that every peer will require at least 3 threads – one to listen on the lookup port, one to listen on the file transfer port, and one to process and display the user interface commands and their output.
7. The UDP socket on the lookup port of a peer can receive:
    a. join requests from other peers,
    b. lookup requests from other peers,
    c. responses to lookup requests generated by this peer,
    d. quit messages from its neighbors.
8. The TCP socket on the file transfer port of a peer can receive a request for a text file located locally on the peer.
9. Try to make each peer robust enough such that if it does not get a response (or gets a bad response) to a TCP or UDP message it has sent, or if one of its neighbors dies, it does not crash but keeps running. It may become disconnected from the P2P network though.
10. You are free to define the format of the specific messages used, as long as the general algorithm described above is followed. In addition, your interface must accept exactly the commands described above in (5).

11. Start work early on this assignment and build it incrementally. For example, first get a peer to join a network; then enable the status functionality; then get lookup to work; then get the file retrieval to work, and finally the quit functionality.
12. Along with your code submission, include a README file that describes how to use your program. I will test your code on Windows 7.

Submit your code and README file zipped up in a single folder to the D2L dropbox `Assignment 3` by the deadline specified.