CSE 4510/5241 HW2
Submit server: course=dc, project=hw2
Due 5pm, Feb 25, 2009

Design and implement a "peer" (estalk), a daemon (estalkd), and the protocol for ESTALK (Extremely Simple Talk) that allows two peers to interactively exchange messages. The daemon serves as the middle-person for setting up the two peers and is **NOT** needed after the estalk peers are ready to communicate directly with each other.

The user commands have these formats:

```
estalk <user>@<host> <local-user> <estalkd-port>
estalkd <estalkd-port>
```

The estalk peer program could "initiate" a talk session or "respond" to one. The "initiator" contacts the remote estalk daemon, supplies information about itself to the daemon, and waits for the "responder" to reply. The "responder" is started by the user after the user is notified by the local daemon; it then gets information from the local daemon about the initiator and starts communicating with the initiator.

An estalk daemon runs on each machine, accepts estalk requests from an initiator, gathers information about the initiator, notifies the local user about the request by printing a message on the screen, (upon a responder's request) provides information about the initiator to the responder, and removes the information.

Since the estalk peer program can either be an initiator or a responder, with the same command format, how does it know which role it is playing?

The messages are synchronized as in the users take turn sending a message. A message can have up to 80 characters and ends with a return key stroke. When the first four letters of a message are "Quit" (from either the initiator or the responder), the program cleans up, sends a "terminate" message to its peer, and terminates. Upon receiving a terminate message, the peer cleans up and terminates.

A session might look like:

```
tarpon% estalkd 55555 &

tarpon% estalk Mary@shark John
[Contacting]
[Ready to talk]
Mary@shark: Hi John, what's up?
John@tarpon: Hi, are you done with the estalk program yet?
Mary@shark: Yes, how about you?
John@tarpon: It's finally done after an all-nighter.
...
John@tarpon: Quit
[Terminating]

-------------------------

shark% estalkd 55555 &
estalkd: estalk request from John at tarpon, respond by "estalk John@tarpon Mary"
shark% estalk John@tarpon Mary
[Ready to talk]
Mary@shark: Hi John, what's up?
John@tarpon: Hi, are you done with the estalk program yet?
Mary@shark: Yes, how about you?
John@tarpon: It's finally done after an all-nighter.
...
John@tarpon: Quit
[Terminating by John]
```

CSE 5241 students only
Additional features:

1. two simultanteous two-way estalk sessions: for example, Mary on host1 talks to John on host2, and at the same time seperately (in another window) to Mike on host3.
2. a three-way estalk session: for example, Kate on host1, Jane on host2, and Mark on host3. When one writes, the other two sees. [For simplification, assume a fixed global order and the users rotate to write: Kate (initiator), Jane (user1), Mark (user2), Kate ...]

```
estalk <user1>@<host1> <user2>@<host2> <local-user> <estalkd-port>
```

What to turn in:

1. Detailed description of your ESFTP protocol (in the README file)
2. Compilation instructions (preferably makefile)
3. Source code
4. Sample session (`script` on unix)

Useful functions:
C/C++: char *cuserid(char *s); int gethostname(char *name, int namelen);
Java: System.getProperty("user.name"); InetAddress localhost = InetAddress.getLocalHost(); localhost.getHostName();