Programming Assignment #2
— Peer-to-Peer (P2P) File Sharing System —

Due on 12/06/2011 (CSCI 4211), Max score: 100pt.

In the first programming assignment you constructed a P2P system and provided functionality for new peer hosts to join an existing host. In addition you implemented the quit command to disconnect hosts from a system. This assignment is an extension of the first programming assignment and will help you understand an application of socket and network programming, namely file sharing and transfer.

You are required to implement a file download service to get files from other peer hosts and a "list" feature to display the list of files in the current system's shared directory. Each host has a bunch of files under its directory that it intends to share with other peers in the system. In order to share these files, a host will have to first search for the desired file amongst other peer hosts, we call this a file lookup service. If the lookup is successful, host downloads this file to its shared directory. Find a detailed description of the File download and Lookup Service below.

What you need to do?

Continue to work on the template you were provided with: peer.c, sockcomm.c and follow the instructions to fill missing parts. Use the compiled binary code 'peer' provided to test how the lookup and download service is supposed to work. It can be executed on SunOS machines. You can follow the example in the later part of this description to see how the program is supposed to behave.

You are not restricted to work on the template provided. As long as your final program has the same function of 'peer' (including output messages), it will be acceptable. Any programming language is acceptable.

Example Setup:

- Step 1: create directories and files on different hosts
  park@cello (~) % mkdir /home/grad03/park/cello
  park@cello (~) % echo "this is mydoc1" > /home/grad03/park/cello/mydoc1
  park@saxophone (~) % mkdir /home/grad03/park/saxophone
  park@saxophone (~) % echo "this is mydoc2" > /home/grad03/park/saxophone/mydoc2
  park@oboe (~) % mkdir /home/grad03/park/oboe
  park@oboe (~) % echo "this is mydoc3" > /home/grad03/park/oboe/mydoc3
  park@trombone (~) % mkdir /home/grad03/park/trombone
  park@trombone (~) % echo "this is mydoc4" > /home/grad03/park/trombone/mydoc4
- step 2: start peer host on 'cello'. Since it is the first peer in the system, that is root, no 'hostname' is specified.
  park@cello (~) % peer /home/grad03/park/cello

- step 3: start peer host on 'saxophone', joining the system thru 'cello'
  park@saxophone (~) % peer /home/grad03/park/saxophone cello.cs.umn.edu
- step 4: start peer host on 'oboe', joining the system thru 'saxophone'
  park@oboe (~) % peer /home/grad03/park/oboe saxophone.cs.umn.edu
- step 5: start peer host on 'trombone', joining the system thru 'saxophone'
  park@trombone (~) % peer /home/grad03/park/trombone saxophone.cs.umn.edu
- step 6: on peer host 'cello', type in command
  get mydoc4
  The file 'mydoc4' is downloaded from 'trombone' to 'cello' and stored in the cwd (current working directory) of the 'peer' program. In my example, you can find the downloaded file at '/home/grad03/park/cello/mydoc4'. The following figure shows how the peer-to-peer system looks like following aforementioned steps.

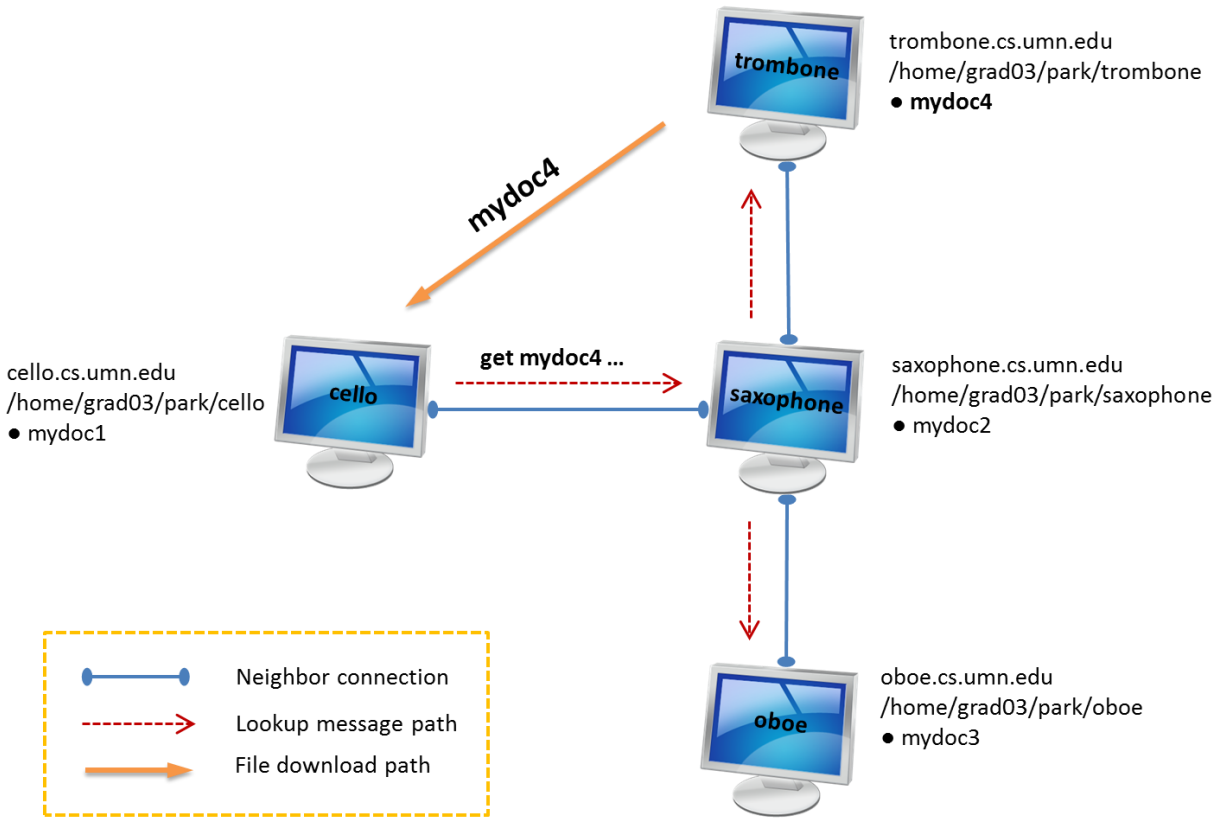

Figure 1: A Tree-based P2P System

Lookup service:

After the peer-to-peer system is set up, a user can type command like 'get mydoc4' on a peer host to download the specified file. This actually involves two steps - lookup and download. The lookup service used in our simple file sharing system is based on flooding. In the previous example, after 'get mydoc4' is input on peer host 'cello', a message looks like 'get mydoc4 128.101.189.164 36953' is constructed. In this message, '128.101.189.164' is the IP address of host 'cello'. '36953' is the port number of the listen socket that 'cello' has created to set up a data channel (or TCP connection) with the peer which has 'mydoc4'.

This message is flooded to all neighbors of 'cello'. For every peer host receiving this message, they first do a local lookup to see whether the specified file is in its shared directory. If it is not there, the peer host forwards the same message to all of its known neighbors, except the one from which it got the message. If the specified file is in a peer host's shared directory, it makes a connection request to the originating peer host whose IP address and port number is in the message (in our example, it should make a connection to host 128.101.189.164 at port 36953). Once the connection is set up, the requested file is sent in multiple IP packets in the case where the file's size is bigger than 1500 bytes.

Download Files considering Timeout:

A new listen socket needs to be created when a 'get' command is received from stdin. Each time, the port number will be different so it is included in the "get" message. Since such listen sockets have timeout requirement, a new process is created using fork(). The new process uses a select() with timeout period, e.g., 10 seconds, to wait for connection requests on the listen socket. In our example, 'cello' creates a listen socket waiting for connection request from some peer host holding 'mydoc4'. However, it's possible that 'mydoc4' doesn't exist in any peer host. To handle this case, such listen socket has a timeout period, for example, 10 seconds. If there is no connection setup after timeout, the listen socket is closed and a message "admin: the requested file doesn't exist in the p2p system" is displayed. On the other hand, when the download is completed, the child process should display a message "the requested file has been downloaded successfully". Like this, this fork structure should allow the user to request a new file while the previously requested files are being downloaded from peers.

Servicing the Files downloaded from Other Peers:

When a peer downloaded a new file from other peer, the file can be provided to other peers which request it. Therefore, you need to update the file list after completing each download. For example, assume that 'saxophone' downloaded 'mydoc4' from 'trombone' and 'oboe' requests 'mydoc4'. Since 'trombone' has 'mydoc4' now, it can provide it for 'oboe' instead of 'trombone'.

Commands you must support in the peer program:

1. **get**: to download a file from other peers. Note that this is neither a built in command nor related to the HTTP get command.

2. **list**: to display file lists in its own shared directory.

3. **quit**: To terminate a peer host. (Already implemented in Project 1)

Milestones:

● 12/06 (Tuesday): You must submit your **make file as well as all three files** (peer.c, sockcomm.c, and sockcomm.h).

► Please combine all those files into tarball (*.tar) or compress them for convenience.

► File Naming convention: Last name + SID. tar (e.g. Park3341123.tar). Please follow this naming convention.

► Please use our on-line submission tool on the class website for your submission.