Task 1 (Napster-style peer-to-peer (P2P) file sharing system)

Purpose: first to get you familiarize with sockets/RPCs/RMIs, processes, threads; second to learn the design and internals of a Napster-style peer-to-peer (P2P) file sharing system.

You need to design a simple P2P system that has two components:

1. **A central indexing server.** This server indexes the contents of all of the peers that register with it. It also provides search facility to peers. In our simple version, you don't need to implement sophisticated searching algorithms; an exact match will be fine.
   Minimally, the server should provide the following interface to the peer clients:
   - registry (peer id, file name, ...) -- invoked by a peer to register all its files with the indexing server. The server then builds the index for the peer. Other sophisticated algorithms such as automatic indexing are not required, but feel free to do whatever is reasonable. You may provide

   optional information to the server to make it more 'real', such as the clients' bandwidth, etc.
   - search (file name) -- this procedure should search the index and return all the matching peers to the requestor.
2. **A peer.** A peer is both a client and a server. As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file. As a server, the peer waits for requests from other peers and sends the requested file when receiving a request.
   Minimally, the peer server should provide the following interface to the peer client:
   - obtain (file name) -- invoked by a peer to download a file from another peer.

Other requirements:

- Both the indexing server and a peer server should be able to accept multiple client requests at the same time. This could be easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.

- No GUIs are required. Simple command line interfaces are fine

Task 3 (nothing new)

What you will submit:

When you have finished implementing the complete your 2 tasks as described above, you should submit your solution on drive link on the course page:

1. **Output file:** A copy of the output generated by running your tasks. Ex: When it downloads a file, have your program print a message "display file 'foo'" (don't print the actual file contents if they are large). When a peer issues a query (lookup) to the indexing server, having your program print the returned results in a nicely formatted manner.
2. **Design Doc:** A separate (typed) design document (named design.pdf or design.txt) of approximately 2-4 pages describing the overall tasks design, and design trade-offs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. **Manual:** A detailed manual describing how the tasks work. The manual should be able to instruct users other than the developer to run the tasks step by step. The manual should contain at least one test case which will generate the output matching the content of the output file you provided in 1.
4. **Verification:** A separate description (named test.pdf or test.txt) of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.
5. **Source code:** All of the source code for all tasks.