# SNAPSTER: A Generic File Sharing System

CS4254: Final Project
Fall 2002
Deadline (Demo): 12/09/2002 – 12/12/2002.
Deadline (Report): Due during your demo time slot

## *Introduction*

Since the common abstraction for data storage is files, generic file transfer protocols form the basis for information sharing and retrieval. In the first two projects, you implemented a generic file transfer protocol that supports multiple client connections. In this project, your generic file transfer protocol will form the basis of a larger file sharing system that allows users to access a distributed data storage system based on simple keyword search. The specification of this project is based on a broad outline, within which you may come up with your own design. In particular, the project specification does not specify the abstraction used for network communication (sockets, RPCs etc), the concurrency mechanism (fork, threads, apparent concurrency etc.) or the underlying transport protocol (UDP/TCP). Your system is similar in principle to the service provided by the now defunct Internet company Napster™.

## *Implementation*

The project consists of two components:

1. A file transfer client and server that runs at each user site. The client allows a user to access files stored at remote user locations. The server is responsible for providing file transfer services requested by a remote client.
2. A centralized server that:
    a. Allows users to "login" and specify the description of shared files from their sites. The login mechanism is used to securely authenticate a user.
    b. Provides a search facility that can be used to perform simple keyword searches. The result of the search is the location of the remote resource.

Figure 1 shows the architecture of the system. Each user host on the system has two components, a file transfer client and a file transfer server. When the file transfer server at a user site starts up, it prompts the user for a username and password and sends this information to the centralized server. Here, you will also need to implement a mechanism to register a user for the first time at the centralized server. Authentication can be provided using the same mechanism as UNIX login/password scheme. Take a look at the man page for the library function `crypt()`. When you authenticate a user, **never** send an unencrypted password over the network.

The centralized server checks the user's identity and sends a success/failure message based on the result of the authentication. If the user is successfully authenticated, the FTP server at the user site uploads (a) the hostname of the user site (b) the names of the shared files and (c) textual descriptions of the shared files. The textual description of a shared file is used for keyword searches to locate a remotely shared file. You may assume that the shared files at the user site are located in a single directory, although more sophisticated implementations would allow a user to share files in a directory as

well as all sub-directories inside it. The shared file directory also contains a "**description file**", which contains the name of the shared file and a textual description of the file's contents. For example, if you have a shared file called MP3.txt, your description may look like:

```
MP3.txt
FAQ on the compression algorithm used in MPEG-2 audio encoding.
```

You may use any file format for the description file. You may also include additional fields in the description, such as author of a file, creation date, file length etc.

You will need to pass the hostname and the port number of the centralized server to both the ftp client and the ftp server at the user site, either on the command line or by supporting a connect command.

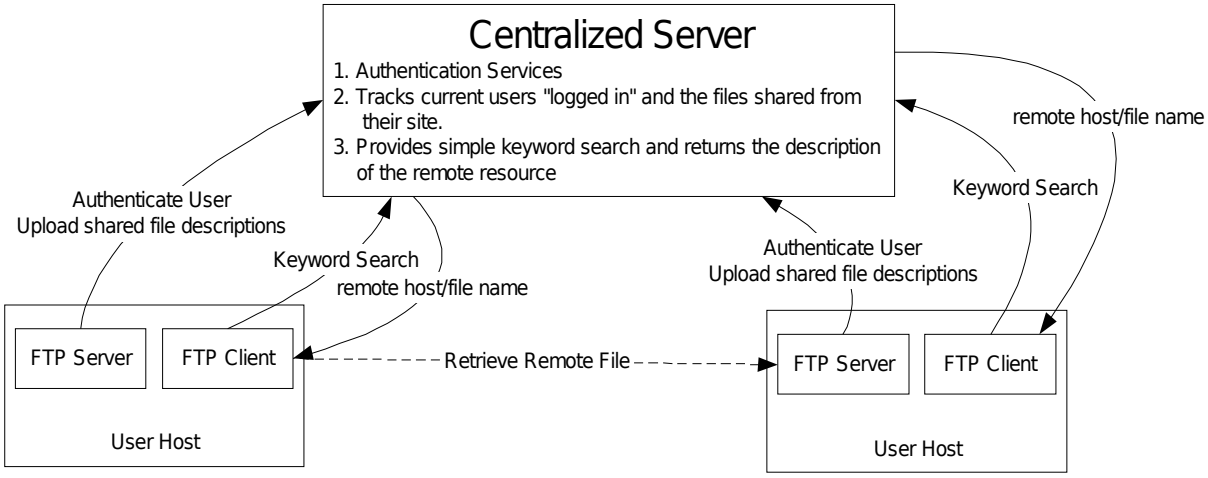


**Figure 1: Architecture of the SNAPSTER file sharing system.**

After the FTP server starts up and uploads the descriptions of the shared files, FTP clients can now query the centralized server for the availability of shared files. To do this, an FTP client prompts the user for a keyword and contacts the centralized server with the keyword. The centralized server performs a simple keyword search on the uploaded file descriptions using the supplied keyword and returns the resource location of the remote files. The resource location includes the remote hostname and the remote file name. You may use exact match for the keyword search, i.e. the keyword supplied by the client should exactly match the description entry at the centralized server for lower credit. More sophisticated keyword search implementations should perform substring matching using the keyword, i.e. if the keyword is a substring of a description of a remote file, it returns a match. Take a look at the man page for the library call **`strstr()`.**

After the centralized server returns the location and name of the remote file, the FTP client can contact the remote FTP server to retrieve the file. Commands sent between the FTP client and the remote FTP server may be in any format, i.e. you can use either ASCII commands like the first two projects, or codes to indicate a command. The FTP client can only be used to “get” a file and not “put” a file. Hence you don’t need to worry about file locking.

Your client and server may also implement mechanisms to sort the results of a keyword search. For instance, when a user registers with the centralized server for the first time, he/she may be prompted for the capacity of the link to the Internet (e.g. modem, Ethernet, T1, T3 etc). This information can be used at the centralized server to sort the matches of a keyword search based on bandwidth available at the remote host.

The centralized server is responsible for (a) authenticating users and (b) performing keyword searches on the uploaded file descriptions based on a user supplied keyword. For authenticating users, the centralized server maintains a list of registered user names and associated encrypted passwords. When a user registers for the first time, the FTP server at the remote site specifies the user name and the encrypted password. The centralized server stores this information in a local password file. The password file can be in any format. When a user registers for the first time, the centralized server may need to check the password file to ensure that the username is not already in use by another user.

After a remote FTP server has authenticated a user and uploaded the shared file names and file descriptions, the centralized server provides a keyword search facility for remote clients. The centralized server should also track the availability of remote resources. For instance, if a remote FTP server “logs out” of the system, the associated file descriptions should be deleted at the centralized server. For simplicity, you may assume that remote FTP servers issue a log out or quit command to the centralized server to indicate that they are no longer sharing files. More sophisticated implementations may require the remote FTP server to periodically contact the centralized server to indicate that they are still alive and sharing files. If the centralized server doesn’t hear from a remote FTP server within some predetermined interval, it may assume that the remote FTP server has been shutdown. A good design will always associate a remote site with its shared files and file descriptions, allowing easy deletion of a remote site when its FTP server stops sharing files.

## *Other Design Issues*

1. Your FTP server at the remote sites should support some form of concurrency to allow multiple clients to connect at the same time.
2. Your centralized server should allow multiple FTP servers to authenticate themselves and upload their descriptions at the same time, i.e. you need a concurrent centralized server. Think of the design issues here. For example, the keyword search operation at the centralized server requires that all concurrent centralized servers see the same set of remote file descriptions over which they perform the search. What happens if you use fork based concurrency?

3. When a client downloads a remote file to the shared directory you should update the description file with the name and description of the newly downloaded file. More sophisticated implementations will also contact the centralized server and upload the new description file.
4. Think of what will happen if the FTP client tries to retrieve a file from a remote FTP server and there is a local file with the same file name but different file description. Hint: Think of renaming the new file.

## *Grading*

Grading will be based on the sophistication of your implementation and ease of use. Your implementation of the FTP client should present a simple text menu of choices representing the keyword search results. For extra credit, your centralized server may perform better fault tolerance, track additional resources to prioritize keyword match results, maintain/store statistics on users and hosts "logged in", present a graphical user interface etc.

## *Submission information*

**NOTE: For this project up to 2 students may pair up to form a group.**

Your code can be implemented in **any language**. You may use **any OS platform** for your code. However, it is your responsibility to ensure that **the Department of Computer Science has machines** running your OS platform, where you can **demonstrate your code**. You should submit the project to chekhov.cs.vt.edu through the usual manner when you are finished with it.

The submission should contain

- Source code
- All binaries

**You also need to turn in a 5-10 page project report on your design and implementation. The report is due during your demo time slot. Your project report should contain the VT ID numbers of the authors.**

**The submission format is tar**. Use the tar program to archive the contents of your project directory as follows:
If your project directory is called `final_project` and it is under your home directory, then go to your home directory and issue the following command:

```
tar cvf filename.tar final_project
```

`filename.tar` now contains the archived version of your project directory.

Each group is allowed 3 submissions with try numbers numbered 1, 2 and 3. Any submissions past 3 will be ignored. The last submission will be graded.

**You need to demonstrate your implementation of the final project. Demonstrations can be scheduled for any time between 9 am to 5 pm between 12/07/2004 and 12/09/2004 (inclusive). Use the sign-up sheet in**

**class or come to my office hours to schedule your demonstration. Do not send your schedule by email.**

Start your project early. There are no extensions for your final project.