**Design Document**

# Introduction

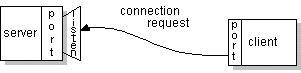
This Software is a Simple Napster Style Peer to Peer File Sharing system and it has mainly two components an index Server Component which maintains a registry of all the Files that have been synchronized with it , and a Client which Syncs with the Index Server and provides users a command Line UI through which users can download the files that are located in a separate Location. (ie a separate Client).

The Implementation of the entire software is done in the Java Programming Language and it’s Implementation of Socket based Communication.

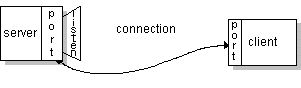
# Sockets

This File sharing program makes use of Sockets entirely for all it’s communication needs. A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. A Socket is essentially a combination of IP Address and Port Number.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

In this Software Socket Implementation can be seen in both the peer side as well as the Index Server Side.

Peer Side -- 2 Sockets are created in the Peer Side – one for connecting to the Index Server to sync with it (this can be seen in ClientAcceptInputThread.java in method connectToIndexServer()) , another Implementation of Sockets called ServerSockets is done for serving the File to other peers to download which is always listening for new connection from other peers(implementation of this can be seen in the Java program ClientServerMultithread.java inside method serveFileForDownload()).

Server Side – 1 Socket is used in Server Side for constantly listening for connection requests from peer.We have used the ServerSocket Implementation here(implementation can be seen in IndexServerSocket.java)

# Multithreading

Since the peer or Index Server in our File Sharing System should be capable of handling multiple requests at a time we make use of what is called Multithreading to make this possible. Multithreading takes care of the multiprocessing and concurrency control part of the project. Threads are essentially Lightweight processes. The ability to execute multiple Threads at a time is called Multithreading. So Multithreading makes it possible to simultaneously execute parts of a program so as to maximize CPU utilization time , and in this case accept multiple connections at a time from multiple clients at a time.

Multithreading is implemented in both the client side and Index Server side here .This is because we need the Index Server to be capable of accepting multiple client(peer) requests for sync and search requests simultaneously , and the client should be able to serve other peer requests the same way.

# Logical Organization Of the System

This system mainly consists of 2 components : (1) Index Server (2) Client (Peer)

(1) Index Server :

The main Roles of the Index Server are to handle synchronization requests from peers or clients and update the local Data Structure (a Map or a Dictionary in our case) based on the information received .The Map will have information such as the File name , ip address and the port number where the File is shared with File Name as the key.The Second role mainly consists of providing Search Functionality where the client requests for location of a file and the Index Server responds by looking into the local Map and sends the information back.

The Index Server is implemented in the following Java Classes :

IndexServerSocket.java : This is the main Class or the Starting point for the Index Server. A lot of the initialization before starting the server is done here like accepting the port from the user for starting the Server and starting the thread which constantly listens for user input.

IndexServerMultithread.java : This class contains the logic for what happens when a request from a client (peer) is accepted .Based on the type of request it performs the corresponding operation which could either be a search operation on the local map or updating the map based on the data received.The entire code is enclosed in a Thread which is concurrently executed based on the request received form the peer or the client.

(2) Client or Peer:

The main roles of the Client or Peer is to Synchronize the Files that it has available for sharing with the Index Server and make those files available for download when a Server requests for download.

The Peer or Client is executed in the following Java Classes:

ClientSocket.java: This is the main class or starting point when starting Client or Peer.It starts the 2 Threads one of which is responsible for accepting input from the user and the other responsible for running the socket which accepts request from other peers.

ClientServerMultiThread.java : This class has logic for accepting file download requests from other peers or clients .The ServerSocket that was created by the ClientSocket accepts the request from the client and serves the request.

ClientAcceptInputThread.java : This class has the logic for displaying the menu to the User and accepting the appropriate input from the user.

TestCasesFile.java : This class is responsible automating some of the tasks required for testing like creating thousand copies of a file in shared directory with different names for the purposes for performance evaluation.

# Interaction Diagram

Data Synchronization & Search

IndexServerSocket

IndexServerMultithread

ClientAcceptInputThread

ClientSocket

showMenu(3,dataMap)

connectToIndexServer(dataMap)

run()

Data Download

ClientServerMultiThread

ClientAcceptInput

ClientSocket

showMenu(3,dataMap)

requestFileForDownload(…)

serveFileForDownload(…)

# Known Risks/Tradeoffs

1. Issue of Scalability when there are a large number of Threads

Since we are creating a new thread every time a new connection is created which is a resource intensive thing to do , when dealing with huge number of requests as is the case when dealing with large scale systems , this approach can present a problem.This approach of using Threads for every connection was used because it is easier to maintain and debug for relatively normal loads.

1. When the size of File is very large it can lead to some inconsistent behavior.
2. Lack of Persistence means that all the details that are saved in the index Server would be lost if the index server is shut down for any reason.

# Features/Characteristics

1. Support for handling multiple requests from multiple clients concurrently.
2. Support for transferring binary files which means files of any type can be transferred from peer to peer.

# Possible Improvements/Extensions

1. Utilizing Non-Blocking mode of communication instead of Blocking and thereby taking away the need for creating an individual Thread for every socket connection.
2. Introducing Persistence to Index Server Side so that the index Map containing Synchronized File Details is not lost even if the Index Server is restarted.
3. Making use of Thread Pools instead of creating a new Thread every time a connection is made. Using worker threads minimizes the overhead due to thread creation. Thread objects use a significant amount of memory, and in a large-scale application, allocating and deallocating many thread objects creates a significant memory management overhead. An important advantage of the fixed thread pool is that applications using it *degrade gracefully.*