**Architecture**

**Server**

The server is a process that starts with creating two threads, Server Listening thread and a Client Listening thread. The main thread then tries to connect with other servers (the addresses have been provided beforehand). It waits for three seconds after sending each server a connection request to go the next one. This is to counter the blocking state that the connection thread will enter since the sockets connect call is blocking. The sockets are made non-blocking before the connect statement and if a connection is formed, they are changed back to non-blocking. The threads run in a non-daemon mode and the Listener threads continue to run where as the main thread exits if it does not form any connections.

Suppose the server which is set up is ‘Server A’. Now let’s suppose another server ‘Server B’ is booted up. It will create its own listener threads and then will try to connect with other servers using the IP addresses that have been previously stored in the server. The Connection thread of Server A will connect with the Server Listener Thread of Server B and a socket will be formed between the two servers. Everything is automatic, only the servers need to be started and the connections will form automatically. After a connection is formed, the two servers will create Server Connection threads to communicate with each other. The client requests are received at the Client connection thread in the server (thread that is created when a client connects to the server). It first validates the client’s commands and then it processes them. It can perform certain operation (create, upload, delete etc.) on the files locally stored on the server and can also request other server to perform operations. Suppose the Client Connection thread receives a request to upload a newer version of an existing file. The Client Connection Thread will upload the file on the current server where the old version exists and send the request to the other servers that contain the file. The other servers will listen to this request through their Server Connection thread.

Each server maintains a Global Directory List or GDL. It contains the names of all files, the version numbers, the file descriptors of the sockets (the server where the file is residing) and the IP addresses of those servers. The Global Directory List is updated firstly at the beginning when the servers boot up. And they are constantly updated when operations are performed on the file. For example, creation of a new file, uploading a newer version file on servers to replace older ones and when deletion is performed. The GDL is prepared beforehand and kept ready, so that when the client requests the directory listing, it is sent as it is. The file name contains the version numbers which are appended to the filename. For example, creating ‘distributed-systems’ will store it as ‘distributed-systems 1’, with 1 representing the version number.

**Client**

The Client is another process, which will be used by the users to request files from the Servers. The Client will request the file from the server and display it to the user. If the user updates the file, it will be sent back to the server, from where it came from. The client starts up with trying to connect with a list of servers. The addresses of the servers have already been stored in the client and this is the first thing that the client accesses. If all servers are down, a relevant message will be displayed to the user. However, if connection is formed, the client will prompt the user for commands to access the distributed file system. The client can ask for the global directory list to check the files that are available. The client can also request the server for files using the download command. Suppose that the client is connected to Server A and the client requests a file. If the file is available with Server A, it will be sent back to the client. However, if the file is with another server suppose ‘Server B’, Server A will redirect the client to that Server B. Server A will return the IP address of Server B to the client so that it can initiate a different connection with that server and request for that specific file. The jumping of the client to re-initiate connections can save the precious time of sending files from one server to another and then to the client. This would also improve the network traffic as least number of messages will be passed.

The client is a sequential program. It waits for the user’s command and then performs that specific task. Since it is sequential, it performs one task at a time.

A screenshot of a cell phone

Description automatically generated

**Semantics**

I have chosen the Upload Download model to access files that are on the servers. The client will request the file by sending the download command to the server. The entire file is downloaded on to the client’s computer. The client can then make changes and on closing, the file is be sent back and uploaded on the server where it came from. If the download file has not been changed, then it is not uploaded onto the server. Other replicas of this file are also updated to prevent from inconsistent data. I have used Session semantics to implement the distributed file system within which changes made within files are not visible to any user other than the one editing it. If the user closes the file, then only it is sent to the servers and from there anyone who downloads it will get the updated data. The problem with this approach was that what would happen if a file is being used by two users. Whose file be saved as the updated one on the servers, if both close the file at the same time. The answer to this is that the file that reaches the last on the server will be considered as the updated one or the final one.

The server design is completely stateless as they do not maintain information about clients. Since no file is opened on the server, there are no file tables maintained and thus there is no limit to the number of files that can be accessed by the clients.

**Single System Image and Replication**

When the client requests a file from the server, the server (to which the client is connected) will look through its global list. If the file resides on that server, it will be sent to the client. However, if the file is located on another server, the IP address of that server is sent back to the client so that it can connect to that server to download the file.

If the client has made changes to the file and closes it, the updated file is uploaded onto the server (that is connected to the client). The server then also updates all other servers that contain the file. This makes sure that the latest file is always available for read.

On Creation the file is uploaded onto the server (that is connected to the client). The server then randomly chooses another server to upload the file for replication. This is to promote reliability and availability by having backups and to allow file access even if a file server is down.

When the servers connect with each other they share their local lists. The servers compare their lists and compare the file version numbers. If a server has a file with an older version number, it requests the other server for that updated file. All of this is done at the startup and the client (user) is completely unaware of these messaging. Once the servers are finished with updating states, they update their global directories. Same is the case with deletion. Suppose Server A and B contains a file named ‘distributed-systems’. If Server A is down and the file is deleted on Server B then there needs to be mechanism to inform Server A about the deleted file. So, when Server A will come live, it will share the local lists of another server and would update its own. It is at this time, it will hide that file ‘distributed-systems’ by appending an asterisk to it. That file will not be displayed on the Global Directory List and to the clients, although it will be physically present on the server’s storage. The next time when all servers are connected, those hidden files will be completely removed from the servers. This delay in deleting these hidden files is to make sure that all servers have marked those files as deleted. And once that is done, they are completely removed.

**Assumptions**

When I was creating this project, I had Satyanarayanan’s Study in mind. This distributed file system that I have created is based upon some of the assumptions that Satyanarayanan made in his study. I am assuming that the files stored are small, less than 20kBs. However larger files can be created and shared between servers (I have done experiments with files around 500kBs as well). Because of this assumption, I have chosen the Upload Download model to transfer entire files between the systems. Another assumption from Satyanarayanan’s study is that file is read more often than they are written to. For this reason, Session Semantics is a great choice for updating files. The downloaded files are only uploaded back to the servers, if they have been edited by the client. If two users are caching the same file, the last one to close it will have his file as the updated one. And for the availability of the files, all created files are replicated on at least two servers.

**Limitations**

The IP addresses of the servers are hard coded in the server and client files. I did not implement the broadcasting method of connecting servers.

**Commands**

These are the list of commands that could be run from the client to request specific operation from the servers.

**ls:** this command is used to list down the Global Contents of the distributed file system. This will list down all files that are currently available. Example use: ls

**refresh:** this command is used to refresh the global directory contents. Example use: refresh

**download:** this command along with the file’s name is used to request the file from the server. When the file is downloaded onto the client, it is opened with a text editor and allows the user to edit it. On closing the file, the file is uploaded back to the server. If the user has not made any changes and closes the file, it will not be uploaded onto the server. Example use: download distributed-file

**create:** this command will create a new file on the client and open it in a text editor. The user can then write to it. The file will be automatically uploaded onto the servers, when the user will close the file. Example use: create distributed-file

**delete:** this command can be used to delete a specific file, located anywhere on the servers. Example use: delete distributed-file