Distributed File System Summary

For my distributed file system I implemented the following components:

1) Distributed Transparent File Access
2) Directory Service
3) Replication
4) Lock Server

I also partially implemented a Security Service.

Distributed Transparent File Access

To enable transparent file access I implemented a client proxy which handles all interactions with other servers via TCP socket connection. The interface for the proxy has the following commands:

1) **openFile(filename)** - Opens a file for editing. Note, “filename” includes the path to the file.
2) **writeFile(filename, data)** - Writes “data” to the location and filename specified in “filename”. This can only be performed when the given file has already been opened.
3) **closeFile(filename)** - Closes the file with path/name “filename”.
4) **readFile(filename)** - Returns the contents of the file at “filename”.
5) **login(username, password)** - Requests a session key and a ticket from the authentication server for use in accessing files. No other actions can be performed until this has been done.

Directory Service

My directory server maps directories to server:directory identifiers. I implemented two directories: “user” and “docs”. When processing read requests, my directory server chooses a copy at random and returns its address to the client proxy. When processing write requests, it sends the client proxy the address of a primary copy.

The directory server also keeps track of the status of each server. If a client proxy notices that a server is unresponsive, it can notify the directory server and the directory server will

mark it as crashed. While a server is marked as crashed, the directory server will not direct any client proxies to it.

Replication

I implemented and primary copy model to handle replication in my solution. After a write operation has been performed on a copy (the primary), the same write command is sent to all replicas known to the primary.

Lock Service

My locking server maintains a list of filenames with associated booleans indicating whether or not a file is currently locked. A file is locked by a client proxy when they open it, and unlocked when they close it. In the case where a file is already locked, the lock server will wait until the file has become unlocked before replying to the client. Built into the client proxy is a separate thread which closes the file after ten seconds, if the client has not done so already. This will prevent the case where a client forgets to close a file, but will not protect against the case where a client crashes with a file open.

Security Service

I have only partially implemented a security service for my system. I used the kerberos protocol for authentication between the client proxy and the authentication server. The encryption I use is extremely basic (concatenating the password onto the end of the message) and was done more as a proof-of-concept than a genuine security measure.

Were I to finish this implementation, I would extend its use to all servers. The client proxy would send a request for a particular service (specified by some globally known ID) to the authentication server, which would return the relevant token. I would also write my own socket implementation to handle the encryption/decryption of messages across all servers.