Distributed File System

Components:

The distributed file system consists of the follow parts:

Directory Server - responsible for storing the mapping between a file path and a logical node, where a logical node consists of a primary and a number of replicas.

Authentication Server - uses 3 key encryption to ensure secure communication between all nodes in the system.

File Server - stores the files for retrieval at a later point. The file server can function in either Primary or Replica mode for the purpose of replication.

Distributed File - this provides location transparency to users of this class by hiding all network operations from the programmer and exposing only Open,Close,Read,Write file functionality.

Authentication Server:

This node maintains a lookup table for identities and their associated keys. When an entity on the network contacts the Authentication server it passes along its identity, a phrase encrypted with its key, and the node it wants to communicate with in the following message:

```
USERNAME: [USERNAME]
ENCRYPTED_PASSPHRASE: [ENCRYPTED_PASSPHRASE]
SERVER_IP: [IP OF NODE TO COMMUNICATE SECURELY WITH]
SERVER_PORT: [PORT OF NODE TO COMMUNICATE SECURELY WITH]
```

The keys used are 128 bit AES keys. If authentication is successful, the Authentication server will then reply with the following token (plaintext below, actual token is encrypted with authenticating user's key):

```
TICKET: [TICKET ONLY NODE CAN DECRYPT]
SESSION_KEY: [SESSION KEY FOR ENCRYPTING COMMUNICATION WITH NODE]
SERVER_IP: [IP OF NODE TO COMMUNICATE SECURELY WITH]
SERVER_PORT: [PORT OF NODE TO COMMUNICATE SECURELY WITH]
EXPIRATION_TIME: [TIME SESSION KEY/TICKET STOPS BEING VALID]
```

This information can now be used to communicate securely with the intended node. Any information that is to be send to the node is first encrypted with the session key and the encrypted information is accompanied by the ticket which the node can decrypt to get the session key. Aside for the initial logon message all messages on the network are encrypted and take the following form:

```
ENCRYPTED_CONTENTS: [CIPHER TEXT]
TICKET: [TICKET FOR NODE]
```

The plain text ticket message takes the form:

```
SESSION_KEY: [KEY RECEIVED MESSAGE IS ENCRYPTED WITH]
EXPIRATION_TIME: [TIME SESSION KEY/TICKET EXPIRES]
```

All of the messages used for the other functionalities are encrypted in the message specified above (encrypted_contents/ ticket). The authentication server is assumed to be stable, if it goes down then communication within the system becomes impossible. The authentication server saves its lookup tables to files on the disk after every update to the table. This allows the state of the authentication server to persist through power cycles.

Directory Server:

The directory server maintains two lookup tables in order to perform its responsibilities. The first is an association between a file path and a logical node and file ID e.g. “path/to/file/one.txt” => Node1/17. Node1 does not represent a physical node but a collection of a primary node and a number of replicas. The second lookup table stores the association between a logical node and a number of IP/port/type triple. e.g. Node1 => 1.1.1.1/8000/PRIMARY. The address of a node returned by the directory server depends on the file operation that is requested, read or write, as well as even distribution of load across nodes. If a write operation is requested then the directory server will always return the primary for that logical node. If a read operation is requested then the directory server will choose randomly between the replicas and return one of those.

How a directory server responds to a file not being present is also dependent upon the operation that is requested. If a read is requested for a file that is not in the directory server then the directory server returns an error. This informs the distributed file that it will have to initialise the file locally as it is a new file in the system. If a write is requested for a new file then a primary is selected at random and that file path is then associated with that logical node.

When a primary comes online it must register with the directory server in order for new writes to be directed towards it. A primary registers with the following message:

```
REGISTER_PRIMARY_IP: [IP]
PORT: [PORT]
```

A replica registering with the directory server must additionally specify the IP/port of its primary:

```
REGISTER_REPLICA_IP: [REPLICAS IP]
PORT: [REPLICAS PORT]
PRIMARY_IP: [PRIMARY IP]
PRIMARY_PORT: [PRIMARY PORT]
```

The directory server pings a replica before returning its details to the user. In this way the system can handle replica node failure as if the replica does not respond then another replica is selected. Just like the authentication server, the directory server also saves its lookup tables to files in order to maintain state after power cycling. The directory server is also assumed to be stable and the failure of this node will prevent file access within the system.

Lookup Request Message:

```
LOOKUP: [PATH TO FILE]
TYPE: [READ/WRITE]
```

Lookup Response Message:

```
LOOKUP_ID: [FILE ID ON NODE]
IP: [IP OF NODE]
PORT: [PORT OF NODE]
```

File Server:

The file server is responsible for the storage and deliver of files to requesting users as well as replication of files for greater uptime and redundancy in the event of failure. When a user writes a file to a file server in primary mode, the primary will open connections with its replicas immediately and write the new/updated file to them. Replicas register with their primary in the same way they do with the directory server. The primary pushes to the replicas as it is the fastest way of maintaining consistency between all nodes in the logical node. The primary is assumed to be stable, as such there are no procedures in place to handle its failure, such as an election between replicas to determine a new primary.

In replica mode the file server responds to read requests from users and writes new/updated files it receives from the primary to its disk. When a replica starts up it is pointed to its primary. The following steps then occur in order to sync the new replica with its primary:

1. Replica registers with primary so that it will be informed of future updates.
2. Replica requests a list of all the file IDs which exist at the primary.
3. It is possible that the replica may have some of these files as it may have gone offline and come back. The simple way to sync would be to request a read of each of these files in turn and save them. This is very inefficient as the replica may have only been offline for 1 second and the primary may have thousands of files so a more efficient system is used. If the replica does not have a file that is on the primary's list then it requests it from the primary. If the replica does have the file it must determine if the file is up to date. To do this the replica requests a hash of the file from the primary, it then performs a hash of its own version of the file. If the hashes match then the file is up to date and there is no need to download it. If the hashes do not match then the replica downloads the file from the primary.
4. Replica can now register with directory server as it is consistent with the primary.

The file servers transfer file by sending read and write file requests that contain the file ID of the file they wish to read/write. In both cases the side sending the file will send the length of the file in the message. Once the message ends to side receiving the file reads that number of bytes and then they have the entire file.

write file request:

```
WRITE_FILE: [ID OF FILE TO WRITE]
LENGTH: [LENGTH OF FILE IN BYTES]
```

Read file request followed by the response containing the length of file before file transmission starts:

```
READ_FILE: [ID OF FILE TO READ]


READING_FILE: [ID OF FILE ABOUT TO BE SENT]
LENGTH: [LENGTH OF FILE IN BYTES]
```

Distributed File:

This system is based on the Andrew File System. On file open the file is downloaded from the file system and held locally. All read and write operations are directed at the locally cached copy. When the file is closed, the updated file is uploaded back to the file system.

Procedure for File Open (i.e. reading file from network assuming file exists):

1. Request ticket for Directory server from Authentication server.
2. Perform file lookup for a read from Directory server, this will return address of a replica node.
3. Request ticket for returned replica node from Authentication System.

4. Request file read from replica node.
5. Received encrypted file from replica and decrypt it.

Procedure for File Close (i.e. writing file to network):

1. Request ticket for Directory server from Authentication Server
2. Perform file lookup for write from Directory server, this will return address of a primary node
3. Request ticket for returned primary node from Authentication system
4. Request file write to primary
5. Encrypt file with session key and send it to primary

Command line parameters for startup:

Authentication Server:

./start.sh [PortToRunOn]

example: ./start.sh 10000

Directory Server:

./start.sh [PortToRunOn] [Username for authentication] [password for authentication] [Ip/host name of auth server] [port of auth server]

example: ./start.sh 10100 rcasey 12345678 localhost 10000

File Server in Primary Mode:

./start.sh [PortToRunOn] [Username for authentication] [password for authentication] [Ip/host name of auth server] [port of auth server] [Ip/host name of directory server] [port of directory server] [type: PRIMARY]

example: ./start.sh 10200 rcasey 12345678 localhost 10000 localhost 10100 PRIMARY

File Server in Replica Mode:

./start.sh [PortToRunOn] [Username for authentication] [password for authentication] [Ip/host name of auth server] [port of auth server] [Ip/host name of directory server] [port of directory server] [type: REPLICA] [Ip/host name of primary] [port of primary]

example: ./start.sh 10201 rcasey 12345678 localhost 10000 localhost 10100 REPLICA localhost 10200