CENTRALIZED PEER TO PEER ENCRYPTED FILE DISTRIBUTED SYSTEM

TEAM MEMBERS

Zaheer Abdul(WF44252)
zaheera1@umbc.edu

Akram Mohammad (OO81170)
akramm1@umbc.edu

Girish Chandra Dama(WS95091)
gdama1@umbc.edu

Sainath Madadi (PB92577)
pb92577@umbc.edu

1. AIM:

The project is aimed to develop Peer to Peer Distributed File System with security.

In this project, a peer successfully will be able to create, update, read, write, delete and restore files on other peers with encryption enabled.

2. Architecture:

The Architecture of this project includes below entities -
Entity 1 : Peer (It acts as both client and server)
Entity 2 : Master Server (Contains all the Information related to all the files).

2.1 Entity 1: Peer:

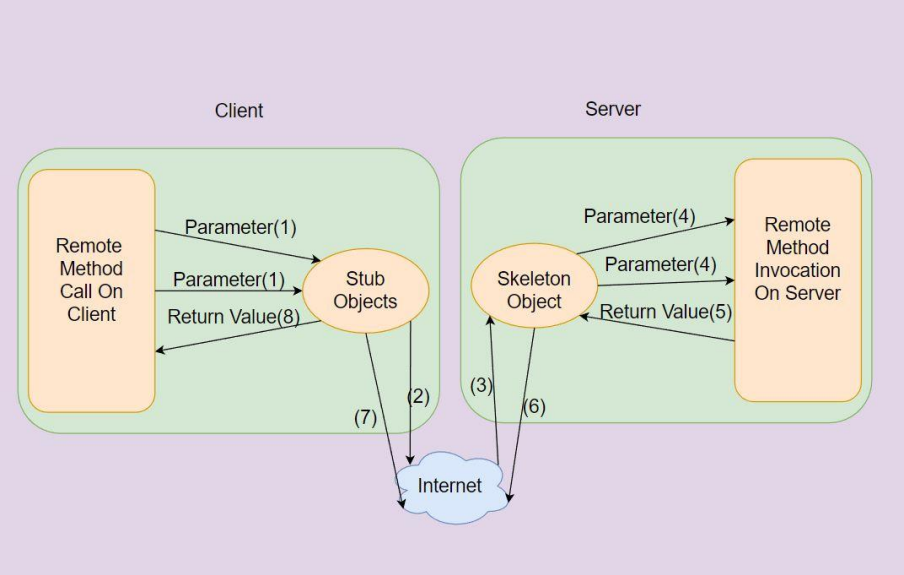
In our Architecture the Peer acts as both Client and Server. The code for client and server was separated and executed individually. The server part handles the incoming requests. The client part enables us to execute create, read, write, update, delete and restore file operations. The key to encrypt and decrypt the file name and file data was shared by Master Server.

2.2. Entity 2: Master Server:

Master Server keeps track of all the files and their corresponding storage locations. It also stores the names of the files that are deleted, secret keys and permissions for each file. It saves the list of users in a Hash Set.

2.3 Design

- The major component of the project is the RMI library. RMI stands for Remote Method Invocation. It is used for all the communications between entities in this project. The communication between peer - master and peer - peer was done using this library. Communication is achieved by invoking the method in a remote object existing on different address space either on the current machine or on the remote machine.



- For detecting malicious activity, the master server checks the existence of each file on corresponding peers at regular intervals using the scheduled executor service, which was defined in java.util.concurrent package. If it is found that the file is deleted or modified, the application exits with the error message.
- Multithreading is used for handling concurrent requests. Since each of the methods on peer require some data to send after the file operation, we have used FutureTask class defined in java.util.concurrent package. It spawns a thread and using the get method we can retrieve the results of the file operation.
- We have used AES encryption and decryption. AES stands for Advanced Encryption Standard. It uses the same key for encryption and decryption. All the data which is transferred is encrypted. We have taken the key size as 128.
- The files are never deleted, for each file there is an entry in the Hashmap and a corresponding boolean value. This value tells whether the file is deleted or not. If the file is deleted, the boolean value for that file will be true and vice versa. To delete a file, change the boolean value from false to true. While the file operations like read, write and update are executed, we first check this boolean value and then corresponding action is taken.
- For managing permissions, we have a Permission class. For each file, we have an object of this permission class. Each object of this class contains Set for read, write and delete. Read set contains a list of peer URI's having read access to this file. Similarly for write and delete. A peer can set and revoke permission for other peers.

2.4 Flow of file operations:

- To create a file, the peer client part first establishes a connection with the master. Master returns the URI's of peers based on replication factor i.e, if replication factor is 3, the master returns URI's of 3 peers, so that the file is created at those given 3 peers. After the current peer receives the list of URI's from the master server, It establishes a connection using the URI and creates the file at all those peers. Master also returns the key to encrypt the file name while returning those URI's. This key is used to encrypt the filename and file data while creating the file at another peer.
- To read a file, the peer client part first establishes a connection with the master. Master initially checks if the peer has read permission to access that file. Master then returns the key and the peer URI of one of the peers containing that file. Peer client then establishes a connection with another peer server and performs a read operation. The file data which is received is decrypted using the same key provided by the master.
- Write operation primarily replaces the contents of a file with new data. For write operation, the peer client connects with the master to get peer URI's and secret key. Masters before returning checks for the write permission of the current peer to that file. Current peer invokes the write operation on all the peers containing that file. While writing the data, it is encrypted using the key provided by the master.
- Update operation is similar to write operation but the new data is appended to the existing file data. The peer client connects with the master to get peer URI's and secret keys. Current peer invokes the update operation on all the peers containing that file. Before sending the data to the update operation it is encrypted using the secret key provided by the master.
- To delete a file, peer first establishes connection with master. If the file exists, then it is deleted.
- To restore a file, peer connects with master and invokes the restore function. The deleted file is then restored and is available for read, write etc., and other file operations.
- To delegate permission, peer connects with master and invokes delegate permission function, and corresponding update is made in the system.

Client: The Client gets connected to Both Master and Peer , At first it connects to master and gets information about the peer and then it connects to the peer by using the RMI (Remote Method Invocation). The client also implements the encryption on the data. For instance, if a client is trying to create a new file , the file is encrypted in the

client itself and then shared with the peer. Similarly when the client tries to read the file from the per it decrypts the file and then accesses the file.

Server:
All the client requests are processed by the server and response of each operation is sent back to the client. As we are sending the encrypted data to the client the server doesn't know any information regarding the data.

Implementation :

MasterQuery:

Read: This function initially checks whether the file is present in the distributed file system. If the file does not exist, it returns <filename> doesn't exist. If it exists, it further checks for the permission of the peer to read that file.If the file does not have permission it return the message "The peer does not have permission to read".
It returns the peer URI containing that file.

Create:
This function initially checks whether the file is present in the distributed file system. If the file exist, it returns <filename> already exist. If it exists, it further checks for the file in the string set , else it creates a new hashset for the file. Now permission is provided to access the file and it is updated into the lookup table.

Delete :
This function initially checks whether the file is present in the distributed file system. If the file does not exist, it returns <filename> doesn't exist. If it exists, it further checks for the permission of the peer to delete that file.If the file does not have permission it return the message "The peer does not have permission to delete/restore".
It updates the String list after the deletion of file.

Update:
This function initially checks whether the file is present in the distributed file system. If the file does not exist, it returns <filename> doesn't exist. If it exists, it further checks for the permission of the peer to update that file.If the file does not have permission it return the message "The peer does not have permission to write".

After updation it return the filedata along with the previous existing data.

Restore:
This function initially checks whether the file is present in the distributed file system. If the file exist, it returns <filename> already exist. If it doesn't exists, it further checks for the permission of the peer to restore that file.If the file does not have permission it return the message “The peer does not have permission to delete/restore”.
It restores the file and updates the string list with the file deletion as false.

Has File:
This function checks whether the file is not deleted and verifies the file in the lookup string set. If the above condition is true, it return “master has <FileName>”

Get Path:
This Function provides the peer with a random path and return the path to that peer.

Get Paths:
This Function checks if the file is present in the file system , if it exists it adds the path to the String set and makes an ArrayList for all the paths.

Register Peer:
This function check is the peer doesnt have any null data or if the peer data is not empty.If the above condition is true ,it further checks if the new data is present in the peer.If false it adds the new data to the peer.

Malicious Check:
This Function checks if the file is present in the peerdata.If any filedata is missing in any peer then it return “Malicious activity detected in the Master Server”.

MasterServer:

The main function in the Master Server updates the properties and loads the properties into the config properties file.It further return the propertiesof master port and masterIP.
It creates a registry for the masterPort and binds the MasterIp and MasterPort.

FDS Query:

Read : reads the file from memory (current path)

Create : creates the file with provided name and data

Update: appended the new data for provided file name

Restore: restores the deleted file

Delete: deletes the file

PeerClient:

Create File:
This Funtion fetches the Ip and Port of a random peer and checks weather the url is null , if yes it returns “<Filename> already exists”. Further it uses a lookup method to find reference of remote objects.

Read File:
This Functions checks if the file is present and return “<filename> does not exists” if the file is not present in the file system.
Further it checks the permissions of the file and return “Peer does not have permission to read” if the peer permission is not available.
If the file data is null or empty then it returns “Failed to Read data”. And if all the conditions are true then it reads the file data and returns the data present in the file.

Update file:
This function first verifies if the file is present or not, if it is not present then it returns “<Filename> doesnt exists”.
Further it checks for the permission to update the file, if the permissions are not available then it return “Peer doesnt have permissionto update the file”.
If the above conditions are satisfied the filedata is updated and return “Successfully updated the <filename> data”.

How to execute in your system

Software :

Java IDE :
Intellij IDEA (Any file editor or IDE is fine)
Java SDK Version -> 11

Libraries :

rmi(Remote Method Invocation) : it allows us to invoke a method on an object that exists in another address space. This is the main library that creates connections between Peers and Master.

Java Concurrent library (java.util.concurrent) : Provides classes and interfaces by using which allows us to make our project run concurrently.

Java.rmi.Naming : used to lookup peer server object based on it's unique path

Java files (java.io.File, java.io.FileWriter) : used to read and create files in the server

javax.crypto : it provides the classes and interfaces for cryptographic operations like encryption, key generations etc ….

Project setup

1. Install the IntelliJ IDE and download the source code the git repository (https://github.com/girish716/P2P_FileSystem)
2. Open the project in Intellij and set up the Java SDK 11, which can be easily configured in IntelliJ.
3. Under resources we have properties file, in which we included the IP and PORT of both Peer Server and Master Server. Update the IP addresses according to your network connection.
4. Now open command line and give command "javac *.java" to compile all the java files
5. Now we have to generate stub files of FDSQuery.java and MasterQuery.java files, this can be done by using "rmic FDSQuery", "rmic MasterQuery"
6. Once stub files are generated we have to start the RMI registry this can be done using "rmiregistry &" command
7. Now we can individually start the Server and Client by following commands in separate terminals. "java PeerServer & java ClientServer".