PROGRAMMING ASSIGNMENT – 4

NOTE: This assignment is NOT a group assignment, but an individual assignment.

1. Objective:

To create a distributed file system for reliable and secure file storage.

2. Background:

A Distributed File System is a client/server-based application that allows client to store and retrieve files on multiple servers. One of the features of Distributed file systems is that each file can be divided into pieces and stored on different servers and can be retrieved even if one server is not active.

3. Assignment Description:

In this assignment one distributed file system client (**DFC)** uploads and downloads to and from 4 distributed file servers (**DFS1**, **DFS2**, **DFS3** and **DFS4)**. DFS stands for distributed file system. The DFS servers are all **running locally on a single machine with different port numbers, for e.g. from 10001 to 10004.**

When a **DFC** wants to upload a file to the 4 **DFS** servers, it first splits the file into 4 equal length pieces P1, P2, P3, P4 (a small length difference is acceptable if the total length cannot be divided by 4). Then the **DFC** groups the 4 pieces into 4 pairs such as (P1, P2), (P2, P3), (P3, P4), (P4, P1). Finally, the **DFC** uploads them onto 4 **DFS** servers. So now the file has redundancy, 1 failed server will not affect the integrity of the file.

Deciding which pairs to upload on which server:

This depends on the MD5 hash value of the file. Let x = MD5HASH(file) % 4

The Table 1 below shows the upload options based on x

x value	DFS1	DFS2	DFS3	DFS4
0	(1,2)	(2,3)	(3,4)	(4,1)
1	(4,1)	(1,2)	(2,3)	(3,4)
2	(3,4)	(4,1)	(1,2)	(2,3)
3	(2,3)	(3,4)	(4,1)	(1,2)

Table 1. How to determine pieces' upload locations.

You can use MD5 or MD5SUM system call or your choice of md5 hash library

DFS servers should be able to identify username and password in clear text. And only provide store and retrieve services if the username and password matches.

a. Functions for DFC:

The client needs to run with the following command

dfc dfc.conf

The configuration file dfc.conf contains the list of DFS server addresses, username and password shown below. The username and password are used to authenticate its identity to **DFS**. Please create your own dfc.conf.

```
Server DFS1 127.0.0.1:10001
Server DFS2 127.0.0.1:10002
Server DFS3 127.0.0.1:10003
Server DFS4 127.0.0.1:10004
Username: Alice
Password: SimplePassword
```

Figure 1. A sample dfc.conf

Inside the **DFC**, it should provide 3 commands **list**, **get** and **put**

1) **list** command inquires what files are stored on **DFS** servers, and print file names under the Username directory on **DFS** servers.

   list command **should also be able to identify** if file pieces on **DFS** servers are enough to reconstruct the original file. If pieces are not enough (means some servers are not available) then "[incomplete]" will be added to the end of the file. For example, suppose that two servers which had parts (3,4) and (4,1) in Table 1 are down. In this case, other two servers would be able to give only parts 1, 2 and 3, not 4. Thus, the DFC cannot reconstruct its file completely. Figure 2 shows that a user types "list" in the DFC and it could retrieve enough number of pieces for the two files text2.txt and image.png while it could not have enough pieces for the file test1.txt.

```
list
test1.txt [incomplete]
test2.txt
image.png
```

Figure 2. LIST command and output example

2) **get** command downloads all available pieces of a file from all available **DFS** servers, if the file can be reconstructed then write the file into your working folder. If not, then print "File is incomplete."
   Here is the **get** command example

```
get 1.txt
```

Figure 3. GET command example

3) **put** command uploads file onto **DFSs** using the scheme that we described in the first page.

```
put 1.txt
```

Figure 4. PUT command example

DFS servers must check the DFC's credentials and must serve requests only if the username and password sent from the DFC match those in dfs.conf. Note that dfs.conf is the configuration file of DFS servers.

The **DFC** should be able to print error messages if it fails in authentication.

b. Function for DFS:

The **DFS** servers need to be run by the follow command:

dfs /DFS1 10001 &

dfs /DFS2 10002 &

dfs /DFS3 10003 &

dfs /DFS4 10004 &

In this assignment we will run all 4 servers locally and use port numbers to distinguish them as shown in Table 2.

Server	Port
DFS1	10001
DFS2	10002
DFS3	10003
DFS4	10004

Table. 2 Server ports

Each **DFS** server should have its own directory named DFS1/ DFS2/ DFS3/ and DFS4/ respectively under your project directory.

Each **DFS** server needs to read /dfs.conf so that it knows all available users and their password.

```
Alice SimplePassword

Bob ComplexPassword
```

Figure 5. Sample dfs.conf file format

1) **USER handling**
   Each LIST, GET and PUT command should be accommodated by a valid user name and password, otherwise **DFS** will send the following error message back "Invalid Username/Password. Please try again."

2) **Directory handling**
   When a valid user request comes in. The **DFS** server will always check if there is a folder named after the username under the DFS's directory. **if there is not, create one and use this to handle all file pieces of this user.**
   For example:
   ./DFS1/Alice/ is Alice's folder and ./ is your project directory.

3) **File pieces handling**

When file pieces arrive, store it in the user's folder and rename it in the following way.

Example:
If piece 2 and 3 of 1.txt is received from Alice at **DFS1**. Then store them at:
./DFS1/Alice/.1.txt.2
./DFS1/Alice/.1.txt.3

./ is your project directory
the "." prefix identifies this is a piece file not a conventional file. The numbered suffix identifies which piece it is.

4) **Implement subfolder on DFS**
Right now DFS handles all files from the same user in one directory. Implement an extra command "MKDIR" within DFC, so that you make subfolders on **DFS**.

MKDIR subfolder
Also try to upgrade **LIST**, **GET** and **PUT** commands so that they can access, download and upload files in sub folders of that user.

For example:
LIST subfolder/
PUT 1.txt subfolder/
GET 1.txt subfolder/

5) **Data encryption**
You encrypt pieces at **DFC** before sending them to **DFS** servers using the password in dfc.conf. Choose your own choice of encryption algorithm. This can be very simple as XOR encryption. You can store data in encrypted format or in a decrypted format. Both approaches are okay.

c. Misc.

1) **Time out or server not available**
A client must try for 1 second to connect to the server. If a DFS server does not respond in 1 second, we consider that server is not available.

2) **Handle multiple connections**

DFS servers should be able to handle simultaneous connections from different **DFC** clients, say Alice and Bob, at the same time.

You can use pthead()/fork()/select() and refer your Assignment 2-3.

d. Evaluation

1. Config files are correctly parsed.
2. Files are correctly put as per the mechanism explained earlier.
3. All the functions (GET, LIST, PUT) are working in a loop seamlessly.
4. Text files, image files, files with different extensions are working correctly.
5. **Reliability through redundancy**
   We will check whether the DFC client still shows the files correctly after killing 1 or 2 servers (kill -9 PID of each server). The expected outcome is to show incomplete files as well as complete files with 'LIST' and 'GET' command.
6. **Privacy through encryption**
   We will check whether the file is readable or not after changing the password in dfc.conf. Also, we will check how two clients with the same ID but with different passwords operate. Expected outcome is that the client without the valid password cannot read the file content. For example, if we try requests with username Alice, and one with correct password and another with incorrect password, it must serve requests with valid password.

This assignment is quite an extension to the PA-1. However, this application must be more robust. Means, to get full points, your code must not hang when we are running commands after one another, it must not stop if one server fails or it reconnects again etc.

Above-mentioned are the subset of test scenarios for this assignment and should be considered as general guidelines. There may be few additional/different test scenarios during interview grading.

e. Extra Credits

1) **Traffic optimization (5 points)**
   In the default GET command it gets all available pieces from all available servers it actually consumes twice of the actual data needed. Find a way to make an upgraded GET command so that it can reduce traffic consumption.

f. Submission requirement

1) Please submit your DFC and DFS codes with all configuration files (dfs.conf and dfc.conf) and README file together in one tar.gz file under the format your_identy_key_PA2.tar.gz
2) Please complete the assignment in the directory where the code runs (relative directory), do not specify absolute directory such as /home/user/Desktop/ etc.
3) Include comments in your codes and try to maintain a clear programming style.

g. Few questions that may arise while implementing:

1. What port numbers to use?
Any port numbers can be used on the server. The client must choose the correct port number of each server as mentioned in the dfc_conf file.

2. Should I compute the md5sum of file name or the content of file?
md5sum must be computed of the content of the file, and NOT the file name.

3. Can I use library or system call for md5sum?
Yes, you can use both.

4.Will there will be only one dfc.conf file ?
There will be one dfc.conf file per User.

5. Will servers have only one dfs.conf file?
For simplicity, you can use the same dfs.conf for all four servers. In real systems, each server running on a different machine runs its own dfs.conf while the content of this file may be the same across different servers.