PROGRAMMING ASSIGNMENT – 2

Due: 10/12 11:59 pm

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 feature of Distributed file system is that each file can be divided in to pieces and stored on different servers.

3. Assignment Description:

In this assignment one client **DFC** (Distributed File Client) is uploading and downloading files onto and from 4 servers **DFS1**, **DFS2**, **DFS3** and **DFS4**. (DFS means Distributed File Server.) In order to be able to run this assignment in a single machine. The DFS servers are all running locally with different port numbers from 10001 to 10004.

When **DFC** want to upload a file to the 4 **DFS** servers, it first split the file in to 4 equal length pieces P1, P2, P3, P4 (a small length difference is acceptable if the total length can not be divided by 4). Then the **DFC** group the 4 pieces in to 4 pairs (P1, P2), (P2, P3), (P3, P4), (P4, P1). At last 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 bellow 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.

Note: when handling MD5 in number format try not overflow
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 need to be 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 show identity to **DFS**, so requests can be accepted. 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. dfc.conf configuration file

Inside the **DFC** client it should provide 3 functions **LIST**, **GET** and **PUT**

1) **LIST** command inquires what file in stored on **DFS** servers, and print file names stored under Username on **DFS** servers (e.g., ./DFS1/Alice, ./DFS2/Alice. ./ is your project directory).

   LIST command should also be able to identify if file pieces on **DFSs** are enough to reconstruct the original file. If pieces are not enough (means some servers are not available) then "[incomplete]" will be added to then end of the file.

   For example:

```
LIST


1.txt
2.txt [incomplete]
3.txt
4.txt
```

Figure 2. LIST command and output example

2) **GET** command downloads all available pieces of a file from all available **DFS**, if the file is reconstructable then write the file into your working folder. If the file is not reconstructable, 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 **DFS** using scheme that we described in the first page.

```
PUT 1.txt
```

Figure 4. PUT command example

Remember to send username and password info to server in clear text to be identified.

Also **DFC** should be able to print error messages send back by **DFS** servers

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 run all 4 servers locally and use port number 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.

After start each **DFS** server need 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 come in. **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.

c. Misc.

1) **Time out**
If DFS server does not response in 1 second, we consider the 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 1.

d. Evaluation

1) **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' command.

2) **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 password operate. Expected outcome is that the client without the valid password cannot read the file content.

3) **Misc. features**

   We will check whether the client optimally choose the servers to download. Also the extra feature such as supporting a subfolder will be tested.

e. Extra Credits

1) **Data encryption (Mandatory for CSCI5273/ECEN5023 students. Extra credits for CSCI4273 students)**
   You encrypt pieces at **DFC** before sending them to **DFS** servers using the password in dfc.conf. Choose your own choice of encryption algorithm.

2) **Traffic optimization (Extra credits for all students)**
   In the default GET command it gets all available pieces from all available servers it actually consumed twice of the actual data needed. Find a way to make an upgraded GET command so that it can reduce traffic consumption.

3) **Implement subfolder on DFS (Mandatory for CSCI5273/ECEN5023 students. Extra credits for CSCI4273 students)**
   Right now DFS handle all files from same user in one directory. Implement an extra commands MKDIR with in DFC, so that you make subfolders on **DFS**.

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

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

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 clear programming style.