Programming Assignment 2

**Announced: Wednesday October 16, 2013.**

**Due: Tuesday October 29, 2013 23:59.**

**CSIEBox**

**Introduction**

In following HW2, HW3, HW4 and HW5, you will build a Dropbox-like service, called CSIEBox. First of all, we build basic functionality in this homework, including uploading files and monitoring the changes on files and directories.

**Architecture**

Server Side

Server has a configuration file, including the path for service root directory and account file. Service root directory is used to store user files on server side. Each user account has its own home directory named after the account name under service root directory. On the server, the account information for all the users are stored in one account file, pointed by the account file path stated above. The following shows an example of CSIEBox home directory and service root directories.

/users/CSIEBox ← Service home directory for CSIEBox service

/users/CSIEBox/b02902xxx ← Service root directory for user b02902xxx

/users/CSIEBox/b02902yyy ← Service root directory for user b02902yyy

Client Side

Each client also has a configuration file, to store server address, account information, and path for local repository.

**Protocol Structure & Actions**

Data structures for server and client communication are defined in provided package. You can find the data structure definitions in **include/csiebox\_common.h**. You can find the example for client and server communication at **csiebox\_client.c::login**.

typedef union {

struct {

uint8\_t magic;

uint8\_t op;

uint8\_t status;

uint16\_t client\_id;

uint32\_t datalen;

} req;

struct {

uint8\_t magic;

uint8\_t op;

uint8\_t status;

uint16\_t client\_id;

uint32\_t datalen;

} res;

uint8\_t bytes[9];

} csiebox\_protocol\_header;

Struct csiebox\_protocol\_header is a base header for communication protocol between client and server, defined in **csiebox\_common.h**. It consists of five attributes. One is attribute Magic, defined for security purpose. Attribute Op defines operation code. Attribute Status defines return status. Attribute Client\_id defines the identification of client processes, which will be used in multiple clients scenario. Attribute Datalen defines the length of optional attributes and will be used to identify the types of optional attributes.

The client/server communication protocols consist of four communication sessions: login, sync file, hard link, and rm. Each client/server communication session has its corresponding data structure to represent the transmitted data. The following illustrate the four communication sessions.

Login session:

Client sends user name and passwd\_hash to server. If the user name and password matches, server returns OK in status field and client\_id. Otherwise, FAIL is returned in status field.

Sync File session:

sync file session consists of two steps in this operation. First, the client sends a sync\_meta request which provides struct stat, file path length and file hash. Then, the client sends file path to server, and server can use path length in sync\_meta header to retrieve the path from a character array. The server checks the file meta and file hash. If file hash on client is different from that on server, the server returns MORE and the client will send file data to server. Otherwise, the server returns OK and terminates the session.

When client receives MORE, client sends the file to the server, using struct csiebox\_protocol\_file.

Hard Link session:

The client sends source path length and target path length to server, followed by source and target path. If a hard link is successfully created, server returns OK in status field. Otherwise, it returns FAIL in status field.

Rm session:

The client sends request path length to server, followed by path data. If the file/directory is successfully removed on the server, the server returns OK in status field. Otherwise, it returns FAIL in status field.

**Requirement**

In this assignment, you will complete the process for linking your local repository to a newly created CSIEbox account. Hence, the process will synchronize the files and directories under client’s local repository, which may or may not be empty, to the server, which is empty. When the process completes, the content of the directories on client side and server must be identical.

The process consists of following steps:

1. (Scan the directory)

The process traverses/scans the local repository directory to find out the existed files and directories, and do the synchronization. In this step, the process starts from the top of local repository and transmits all the files and directories to the server side. You will use the provided socket communication template to transmit the files. However, you have to handle the operations for maintaining directories including creating directories, reconstructing soft/hard link on the server sides, etc.

2. (Monitor the changes on files, directories, hard link, and soft link on local repository): To keep the file/directories to be up-to-date with the local repository, the process has to monitor if there is any change on files/directories. For this purpose, you can use inotify API, which can monitor file system events, to learn the update and conduct corresponding operations. Example program of using inotify API is available in the provided package.

Note that you have to pay special attention on handling the changes on links. In the case of symbolic link, you need to update the value of link instead of updating the content it points to. Even if the file which the symbolic link points to is out of the local repository, or it is a bad link, or there is a loop in the symbolic links, we should do the synchronization anyway. A hard link should be processed in two cases. If it links to a file within the directory tree of local repository, it should remain to be a hard link on server side. Otherwise, it should be treated as a regular file, rather than a hard link.

**Grading**

This assignment counts eight points in final grade. The assignment will be graded according to the following criteria.

1. 5 points for synchronizing existed files and directories between client and server. It includes traversing directory tree on client’s side, and transmit the file and directories to server.

* (1 point) traverse directory tree: output the longest path on local repository, starting from service root directory, to file ***longestPath.txt*** in service root directory. For example, path ***cdir/abcdefghijk*** is longer than path ***cdir/a/b/c***.Therefore, ***abcdefghijk*** is output to ***longestPath.txt***. There is only one longest path in test data.
* (1 point) handling regular file on server side
* (1 point) handling directory on server side
* (1 point) handling soft link on server side
* (1 point) handling hard link on server side

2. 2 points for synchronizing the files, including regular files, directories, and links, ***newly*** created or modified under local repository to server. This and following requirements needs to use inotify API, which can monitor file system update events and notify the process. (0.5 points for each type of files.)

3. 1 point for synchronizing the deleted files and ***empty*** directories under local repository to server.

**Late assignment**: Only the assignments submitted before the deadline will receive the full credit. 5% of your credits will be deducted for every single day delay.

**Plagiarism**: There is absolutely **NO** tolerance for plagiarism. Your codes will be checked by MOSS ([http://www.cs.berkeley.edu/~aiken/moss.html](http://www.cs.berkeley.edu/%7Eaiken/moss.html)).

**Contact**

* Email: [box0922480107@gmail.com](mailto:box0922480107@gmail.com)
* Ptt2 : SysProgram, TA account: f771213