COMP 416 – Computer Networks

Project #1

Due: October 22, 2015, Thursday, 11.59pm (Late submissions will not be accepted.)

Submit your project deliverables to folder:
F:\COURSES\UGRADS\COMP416\HOMEWORK\PROJECT1
Note: You can work in groups of 2 students.

iSync: Development of a basic cloud storage protocol

This project work is about the **application layer** of the network protocol stack. It involves application layer software development, client/server protocol, application layer protocol principles, socket programming and multithreading.

In this project, you are asked to develop a basic cloud data storage protocol named **iSync** based on the client/server model. iSync server uses two or more TCP connections to interact with an iSync client: One connection for sending the protocol commands, and the other connection(s) for data transfers. Fig.1 shows connections for a sample iSync client and server. This approach is a simpler version of the practical cloud data storages such as Dropbox, Google Drive, and Box.

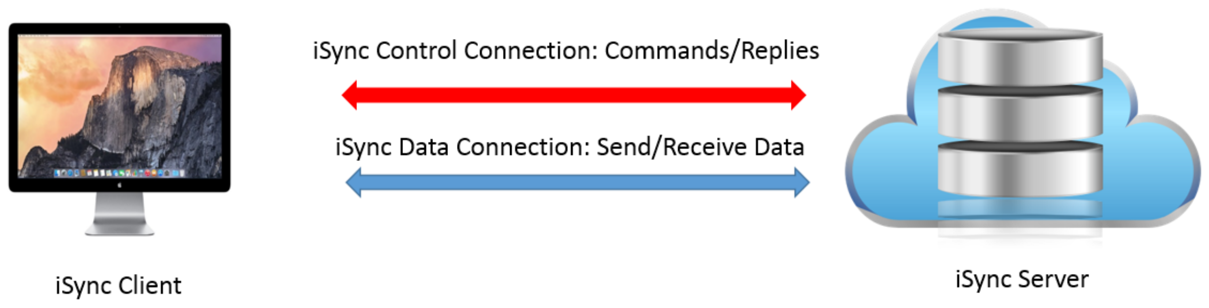
iSync Control Connection: Commands/Replies
iSync Data Connection: Send/Receive Data
iSync Client
iSync Server

Figure 1. Two or more connections for each iSync data transfer query

iSync design has the following main components:

- Server side of the protocol
- Client side of the protocol and commands

iSync should support both text and binary data storage and transfers. We suggest you to start with implementation of text data operations. After that, you can implement binary data operations functionality.

iSync server side requirements:

iSync server should:

- Use predefined unused port numbers for its control and data connections based on TCP sockets.
- Create a new thread per client to server the client's requests. Thus, iSync protocol server side should be multithreaded to support multiple clients. To implement this part, iSync server should have a class as a thread to handle the connection between the client and server. As described in the PS, the server listens the line by the accept() function on a server socket. When a connection is established between a client and a server, the server generates a TCP socket for that client and starts the thread by passing the generated socket. In this way, multiple clients can query the server concurrently on the same port.
- Receive connection requests from clients, receive command requests, and process them.
- Have its storage folder to put files transferred.
- Provide downloading of both ASCII Text and Binary files.
- Sends the hash value of each of its files to the client upon its request.

iSync client side requirements and protocol commands:

iSync client should:

- Have its local client folder to receive the files from iSync server.
- Check the server repeatedly to find the updated files via the command connection.
- Ask the server for the updated files.
- Download the updated files from the server via the data connection line.
- Remove the deleted files in the server side from its local storage.

iSync client is expected to execute the following commands and display their reply messages through its user interface (console). When command is entered by the user, the iSync client side has to send the command and necessary information via the control connection to the iSync server, open data connection and wait for the response from the server. As soon as the response is received, iSync client has to display it on the interface:

- **`sync check`**: This will ask the server for the hash of its files one by one. After receiving the hashes from the server, the client will compare the hash of each server file with its own corresponding file and display a message about the sync status of its files:
  - "No update is needed. Your storage is already updated" or
  - "You have to update your storage with the following operations:

    -<name of the updated file> <should be added/updated/deleted><size>

Example:

```
>sync check
You have to update your storage with the following files:
a.mp3  delete   2MB
b.doc   update  20KB
c.pdf    add       1MB
The total size of the updates is 3.2 MB
Sync check finished.
```

- **`sync all`**: This command is used for synchronizing all the client side files with the server files. In other words, after executing this command, the client local storage folder would contain the same copy of the server storage files. After the client sends this command to the server, the server starts sending the files to be added or updated at the client local storage. One important point about implementation of this command is the efficiency of the sync operation. For example, **it is totally inefficient to remove all the files at the client side and download everything from the server.** Rather, a sync all operation should only download the updated or new files from the server, replace the updated files with their old version. The sync all operation should also delete the files that are already removed in the server side, from the client storage folder. During the execution of the sync all command, the client should show the log of the files that are in the download progress and the files that are downloaded.

  Example:

  ```
  >sync all
  Start syncing
  Deleting a.mp3 2MB
  a.mp3 has been deleted successfully.
  Updating b.doc 20KB
  b.doc has been updated successfully.
  Downloading c.pdf 1MB
  c.pdf has been downloaded completely.
  Sync all finished.
  ```

- **`sync <file name>`**: This command is similar to the command **`sync all`** except that instead of synchronizing all the files, this command performs a sync just for the file given in its argument. If the <file name> does not exist, the client should display an appropriate message. Note that, the client sends this command to the server only for the case of updating or adding a file. In the case that a file needs to be deleted (like the file a.mp3 in the above example), the client will execute this command simply by just removing the file from its storage folder. Same as the sync all, for this command, client needs to show the logs on its interface.

iSync system execution scenario requirements:

Requirements of an example iSync system execution scenario are described below. In the project demonstration, you will be asked to show an execution scenario with the given requirements.

1. There are two separate computers connected to the Internet.
2. One computer would execute iSync client and the other would execute iSync server.
3. iSync server stores all files in a folder entitled "Storage" at the server.
4. There may exist some sample text and binary files (e.g., .txt, .mp3, .mkv) in iSync server Storage.
5. iSync system should support both Text and Binary data transfers.
6. All files that iSync client stores is in a folder entitled "Local" at the client.
7. iSync client should check for the server files' that it does not have or the ones that are updated by the server.
8. iSync client should check for the newly added or updated files at the server side.
9. iSync client should perform downloads from iSync server using one data connection line dedicated to that client (by a thread) and replace the updated files and remove the deleted files.
10. You may be asked to run multiple concurrent clients that query the server.

Important notes:

For the sake of simplicity, you can put a limit on the number of files (e.g. 10 files) that iSync server can host. The server can host up to 10 files, and can update them anytime.

The server merely updates the content of the files, and size of the files may therefore be changed after an update. The duty of the server is to send the hash value for each of its files to the client, and send the files that the client requests.

Hash Tree Usage (Bonus part):

Having a limited number of files in the server side makes the procedure of checking the freshness of the client local storage very easy by comparing the hash values of the files in the client and server side one by one. However, this is not an efficient way of doing that. Asymptotically, having n files in the server side, it takes O(n) time to check the freshness of the client side files. An efficient solution is to use a **hash tree** instead in both client and server sides, and compare the root hash values of the client and server to check the freshness. Figure 2 shows a sample structure of a hash tree.

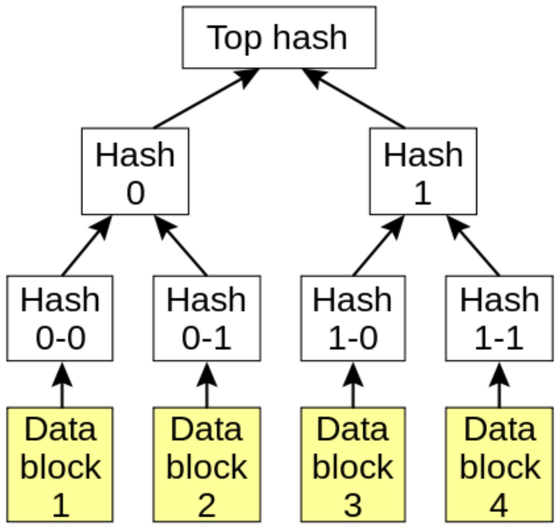

Figure 2. A hash tree for 4 data files

In a hash tree, the leaves are the hashes of the data files. The value of each node except the leaves is the hash of its children. The top hash (root hash) is the hash value associated with all the files. Using a hash tree, regardless of the number of the files in the client and server side, client only checks the root value of itself with the server. This makes the iSync protocol much more efficient especially in the case where there are no file updates for a long time. To implement this part, your system should satisfy the following criteria:

1. Implementing a hash tree and comparing the root values of the client and server.
2. For the case of an update in the server, the client side program should be able to find the place of inconsistency and update the inconsistent files with their latest version.
3. In your report, you are expected to provide a discussion about your hash tree architecture, your hash tree implementation, your inconsistency detection algorithm (all in pseudocodes).
4. Run your program for a large number of big files (e.g., 1 MB, 1 GB, 10 GB) and generate the graph of the update check running time for the case you use the hash tree and the case you do not.
5. During the demo session, in order to test this part, we may ask you to run the server with a large set of files (e.g. 100). For the case that there is no update in the server side, your program should be able to check the freshness in a short time (few seconds).

Project deliverables:

You should submit your source code and project report (in a single .rar or .zip file).

- Source Code: A .zip or .rar file that contains all the source codes with comments.

- Report: in .pdf format describing the design structure and results:

  - Problem description and system components
    - An overview of the classes you designed
    - The relation between the classes
    - What is the flow of your program based on the classes
  - Describe iSync protocol commands implemented:
    - The classes you defined: Their duties and members
    - The primary functions you defined: Their input/outputs and behavior
    - How you implemented the sync operations between the server and client
  - Important points in the implementation
    - How did you use hash functions in your code: rewrite your function call here
    - How did you send a file via a socket: rewrite your function call here
  - Results and conclusions

Demonstration:

You are required to demonstrate the execution of your iSync protocol with the defined requirements. Your demo sessions will be announced by the TAs.

Notes:

1. Please read this project document carefully before starting your design and implementation.
2. In your implementation, do not rename/change any command, folder name and such in iSync protocol specification.
3. In case you use some code parts from the PS codes or the Internet, you must provide the references in your report (such as web links) and explicitly define the parts that you used.
4. You should **not** share your code or ideas with other project groups. Please be aware of the KU Statement on Academic Honesty.
5. For project demonstration, you are supposed to bring two laptops (if possible), and run your project on your own laptops (iSync client laptop and iSync server laptop).
6. In the project demonstration, the task sharing between members of a group should be clear and each member will be expected to answer questions about the project tasks.
7. Your report must present your protocol design aspects and act as a reference for your implementation.