CS456 – Project Final Report

IDENTIFICATION

Janson Webster – 20277029 – jswebste@uwaterloo.ca
Kuen Ching – 20273716 – tkching@uwaterloo.ca

ARCHITECTURE

Server

The server can be run on any machine supporting Java, but the machine must have a static IP address, and the client's must be configured to connect to this ip address. As a static IP address is required, running the server code on a stationary server makes the most sense.

The server opens a socket on the local machine and listens on port 62009. When a client connects to the socket, the server will accept the connection request and then spawn a worker thread to handle communication with the client. This way the main server thread is able to accept multiple connections from numerous clients at the same time without blocking to service any of them.

The server first requires the user to authenticate before any request can be made. The only exception to this rule is when the user sends a registration request described later. Provided with credentials, the server does a database lookup on the provided username to ensure existence. If the username does not exist or the user's provided password is incorrect a rejection message is sent to the user. But in this case the database is also updated to reflect that an invalid login was made by incrementing the number of failed login attempts for the given username. If the number of failed login attempts ever reaches 3, the user becomes locked out and must contact a super-user to have their account reset. This is a security procedure put in place to prevent people's accounts being hacked via a brute force attack. If the user's credentials check out, and they are not locked out, the user's failed login counter is reset and the user is authenticated. It is at this point the user can make any of the following requests.

Possible Requests:

User Registration – This is the only request which does not require the user to login before calling it. When the server receives this request, it first does a lookup in the database on the username provided to ensure that it has not already been taken. Next, the server will create a new database entry in the USERS table for the new user using the provided username and password. Lastly a home directory is created in the root_upload directory on the server. This home directory will always be the same name as the username in full capitals.

User Password Change – This request is used when the user wishes to change their password. This request will only succeed if the old password provided matches the current password of the username. If this check passes, the password associated with the user is updated to the new password.

File Upload – This request is called whenever the user wants to upload a file to the server. All files are uploaded to the user's home directory. This version of the server supports file storage in subfolders located in the user's directory. A caveat to this is that the user must ask a server admin to create any subfolders in their home directory they require. There is no way to do this remotely from the phone itself.

The server first ensures that the file they are uploading does not already exist on the server. If it does, the request will fail and the client will be notified. Next the server will check to see if any partial upload of this file exists. A partial file exists if a file exists on the server with the same name as the one provided, but has a ".part" extension. Also in the database, this file will have its FILES.COMPLETE field set to 'N'. In this case the last_modified timestamp of the file being uploaded and the ".part" file are compared. If they differ, the server's partial copy is stale and must be deleted for the file uploaded. If the last_modified timestamps are equivalent, the file upload can be resumed from where it last left off, at which point the client send the rest of the file to the server. Otherwise, the full file is sent to the server. The server receives the file in byte chunks from the socket and writes them to the output file on the server. When the upload begins, the file is added to the database if it is a new upload, or the file is updated in the database if it already exists. A part file is created whenever an upload is in progress when the connection to the server on either the client or server is interrupted, and thus the connection between the two is dropped.

Once the file has been completely uploaded, the ".part" extension on the file is removed, and the database entry for the file is updated to reflect this change. The file's FILES.IS_COMPLETED is also set to 'Y' at this time.

File Download – This request allows the user to download a file from the server to their local SD cards. Files which are eligible to be downloaded are anything from their home directory (shared or not) which is completed. Users are also allowed to download other user's files if that file is complete and the user has shared their file (FILES.SHARED = 'Y')

The user first specifies the file they want to download and the location in the file they want the server to start sending from (for partial file download explained in the Middle Layer section). The server verifies the user has the privilege to download that file and also ensures that the file is not marked for deletion. Please see the File Deletion section for more information regarding this. The server then sends the data in byte chunks over the socket. Once completed the server will wait for the client to finish downloading the file and then close the connection

Remote File Download – This request allows the user to provide the server with the URL of a file they want the server to download for them. The server will take the URL provided and will download it to a file specified by the user in their home directory. It will then tell the user that the request was successful and it will begin to download the file. There is no way from the server side to let the client know when the download is completed, and thus it is up to the user UI code to periodically check.

File Deletion – This request allows the user to delete a file that they own. The server will first verify that the user is in fact the owner of the file and then it will mark the file for deletion. The real deletion will occur once the last server thread accessing the file exits. For more information about this please reason Issue #3

File List Retrieval – The request allows the user to retrieve the names of all files and folders located in the user provided root directory that they have access too. The server executes a database query for all filenames which start with the provided root and for which the user has the proper privileges for. This means the user must either own the file, or the owner must share the

file and the file must be completed. No matter who owns the file, if the file is marked as deleted it will not be returned. The server then takes the results returned form the query and creates file wrappers for each returned file. This file wrapper stores the name of the file, or if it is a directory, it will just display the name of the directory at the current root directory level. If a file is not completed, a special flag will be set telling the user that this file can only be deleted or appended to via upload. These results are returned to the middle layer by writing objects to the socket stream.

File Existence – This request allows the user to ask whether a given file exists. The server will return true if the file exists and is either owned by the user. It will also return true if the file is owned by another user and is shared and completed. Otherwise the server will return that the file does not exist.

File Permission Change – This request allows the user change whether or not other users are allowed to view and download a given file. The server first ensures the file exists and the user is the owner of it. If this passes, it will update the FILES.SHARED field to either 'Y' or 'N' depending on what the user specifies.

After the request is completed, whether it be a success, failure, or delayed response, the client will then disconnect from the server. If the client wants to make another request they must establish a new connection to the server and go through the process specified above once again. The server is implemented in this fashion for a few reasons. First, removes the need to establish and maintain a persistent long-term connection, which is difficult and expensive. It also ensures that users who are not currently using the service are not holding onto valuable system resources and thus taking them away from other potential users. Once the client disconnects the thread which was spawned to service the user's requests is destroyed.

Database

We have implemented a very simple database to aid in our different verification requirements. There are two housed by the database: a USERS table and a FILES table.

The USERS table has the following fields:
USERNAME – This field stores the different usernames registered in the system. All letters in the username are stored in uppercase to aid in uniqueness and comparison. This field is the primary key of the table.
PASSWORD – This field stores the password associated with a given username.
NUM_FAIL – This field stores the number of times a failed login attempt has occurred for the given user.
IS_LOCKED – This field stores either a ' Y' or 'N' character signifying whether the given user is locked out of the system. This lockout occurs once the NUM_FAIL field reaches a certain number.

The FILES table has the following fields:
FILE_PATH – This is the file path (minus the root directory) of every file which has been uploaded on the server. All the letters in the file path are stored as uppercase to aid in uniqueness and comparison. This field is the primary key of the table. The full file path is stored as this

allows users to have a file with the same name in numerous directories (or for that matter, different users having the same file names)
OWNER – This is the owner of the file. A foreign key constraint exists for this field and it relates to USERS.USERNAME. This is in place to ensure that any file in the system must belong to one real user.
SHARED – This field stores either a ' Y' or 'N' character signifying whether or not the file is shared with other users. If 'Y', all other users can view this file and download it. If 'N', only the owner will know of this file's existence.
COMPLETE – This field stores either a ' Y' or 'N' character signifying whether or not the file has been completely uploaded onto the server. Files which have their COMPLETED field set to 'N' cannot be downloaded, and will not show up when other users view your files, regardless of whether or not the file's SHARED field is set to 'Y'
MARKED_FOR_DELETION – This field stores either a ' Y' or 'N' character signifying whether or not the owner wants the file to be deleted. When this field is set to 'Y', no users are allowed to download this file, and the last user who is currently accessing the file will then delete the file once they are finished.

Middle Layer
All requests made by the middle layer are sent directly to the server via a socket connection. Any errors which may occur during the request are caught here and are propagated to the UI in nice Exception types for easy parsing. A middle layer remote procedure call is successful if it does not throw an exception.

The only complicated request in the middle ware is the download request. This download request is handled very similar to the upload request by the server. This means that the middle layer will first check if the file already exists and fail if it does. Otherwise it will look for a part file and ensure that if it exists, its last_modified date is the same as the file's on the server. If the dates are different, the request will fail. Otherwise the middle layer will tell the server from what point to start sending the data and it will append this data to the partial file. When the file is downloaded, it is received in chunks and saved to a ".part" file until the file is completely downloaded. At this time the ".part" extension is dropped.

UI
Every unique action (Login, Registration, Account Management, Server Management, Remote Download, File Upload) is implemented as its own activity. The user is first presented with a login screen with the option to register. Once logged in, the main screen contains links to the Account Management, Server Management, Remote Download, and File Upload screens. Error messages as well as success messages are propagated to whatever screen the user is currently on, even if the message does not pertain to what is displayed on the screen. This allows the user to continue to use the application while a request is being serviced. To solve the issue described iin the server request for Remote Download (user notification), the UI spawns a thread for every Remote Download request which periodically makes a File Existence request to the server as to whether the file being downloaded is finished. When it is it will let the user know and shut down.

COMMUNICATION PROTOCOLS

Authentication – The user sends a greeting message to the server with its username and password. The server then verifies the credentials and sends back: a bad authentication message if the

credentials are bad, a locked out message if the user is locked out, a greeting message to signify that the user is authenticated

User Registration –The user sends a registration request message to the server with their desired username and password. The server will send back: a registration failed message if a server error occurred during handling the request, a registration invalid message if the username is already taken, a registration ok message if the request was successful

User Password Change – The user sends a password change request message to the server with their username, old password, and new password. The server will send back: a password change failed message if the old password provided was incorrect, a password change ok message if the request was successful.

File Upload – The user sends an upload request message to the server with the path the file is to be stored at on the server, the size of the file being uploaded, whether the file is to be shared, and what the last modified date is on it. The server will send back: an upload reject message if the file already exists or any other general error occurs, an upload out of date message if the part file on the server is out of date, an upload ok message with the byte number to start the upload at if the server is ready to accept the file, an upload finished message if the upload was successful.

File Download – The user sends a download request message to the server with the path to the file, and the owner of the file. The server will send back: a download reject message if the user does not have the required privileges to download the file, a download ok message is sent with the size of the file and the last modified date if the server will accept the request. After sending a download ok message, the server waits for the client to send a download ok message back before sending the file. After sending the file, the server waits for a download finished message to be sent from the client to say that the download was completed.

Remote File Download – The user sends a remote download request to the server with the URL to download, the location to put the file on the server, and whether the file is to be shared. The server will send back: a remote download decline if the file already exists on the server, a remote download accept if the server is downloading the file.

File Deletion – The user sends a delete file request to the server with the file to be deleted. The server sends back: a delete fail message if the file does not exist or the user does not have the correct privileges, a delete delayed message if the file was marked for deletion but not deleted, a delete success message if the file was successfully deleted.

File List Retrieval – The user sends a file list request to the server with the root directory. The server will send back: a file list fail message if an error occurs during the request, a file list success message if the list was successfully generated and sent.

File Existence – The user sends a file exists request to the server with the file path and the owner. The server will send back: a file exists rejects message if the user does not provide enough arguments for the request, a file exists no message if the file does not exists or the user does not have sufficient privileges to the file, a file exists yes message if the file exists and the user has sufficient privileges.

File Permission Change – The user sends a permission change request to the server with the file path of the file and the new privilege level. The server will send back: a permission change fail message if the user does not own the file or the file is incomplete, a permission change success message if the file's permission level was successfully changed.

ISSUES

The following are some of the issues we faced during development and how re resolved them.

1) We needed a way in which to store large amounts of data pertaining to usernames/password as well as file permissions. This method had to be easily updatable as this information could be changed at any time. This ruled out storing the data in a csv file as maintaining the file would be time consuming and the file would also be prone to corruption. This is why we chose to create a basic database in which we could store our data. We chose to use Oracle as we have both had experience creating and using this database system.
2) An issue arose when a user wants to delete a file which is currently being downloaded by other users. Obviously we can't kill the session with the other users, but we also don't want to force the user wanting to delete the file to have to find a time where no one is downloading the file. To resolve the file we decided to implement a postponed deletion system. When the user requests the deletion of a file, the file will be marked for deletion. After this point no more download requests for this file will be fulfilled. Once the last server thread accessing the file (which could very well be the user requesting the deletion) is done using the file, it is responsible for purging the file from the system.

TESTING

The server code was tested alongside the middle layer code at the same time by calling all remote procedure calls from a test client. For each of these calls, all failure cases were tested as well as all success cases. The following details some of the more advanced test cases we ran.

1) Uploading/Downloading a file which already exists as a .part file. Both the case where the last_modified date was unchanged (success), and the case where the last_modified date was changed (failure) were tested.
2) Ensuring that a file deletion request which occurs while other users are actively downloading the file is postponed until the last active user is finished with the file. We also ensured that no more users are able to see/download this file once the deletion request is made.
3) The clients properly error when they lose internet connectivity.
4) Multiple clients are able to concurrently make requests and access the server, and the main server thread does not block during this time.

We also tested the application again one the UI was hooked in to ensure all of the middle layer hooks worked properly, and errors/success message were properly propagated to the user. We also ensured that the user is able to make multiple concurrent requests to the server from their device without issues.