CT213 Assignment - BashBook

1. By Conor Salmon (22402456) and Stephen Murphy (22462694)

2. This is the report of our BashBook project, which makes use of Bash to create a Facebook-like app that creates users with friend lists and walls, where friends can be added, and comments can be left on walls.

3. Starting with the scripts from Part 1 of the assignment, we have the create.sh script, which takes in $id as a parameter. This $id will be the name of the user, and this script creates a directory, named after the user, and containing two .txt files, friends and wall, which we will use in the following scripts. It will also check to see if a user with that name already exists, returning an error message if so.

The next one we created, add_friends.sh, takes two users as parameters, $id and $friend. This code will add the name of $friend to the "friends.txt" file in the $id directory. This script also checks to see if both usernames entered are those of real users and will also check to see if they are already friends, in which case they don't need to add each other a second time.

The third script, post_messages.sh, takes in three parameters: $id, $rec_id and $wall_message. The purpose of these three are: one who sends a message and the other who receives it respectively, and the third parameter will be that message. As with add_friends.sh, this script will check to see if both users exist, and it will also check to see if the two users are friends, by reading the friends.txt files from both users. If all of these conditions are met, then the $message will be added to the wall.txt file in the $receiver's directory. The message is preceded by "$id says: " so the receiver can see who sent the message to their wall.

Next, we implemented a way to view a user's wall. This was done using the display_wall.sh script, which needs one user $id for the wall being displayed and $requester for the user requesting to look at it. This script will take in and print out the contents of the wall.txt file inside whatever directory is named $id. As before, the script checks to see if the user exists before running the rest of its code.

After that, we created the server.sh script, which is a script from which the previous four could be executed. This is done with an endless while true loop where the user is repeatedly

prompted to enter a command, which are simplified versions of the above scripts' names, such as "post" instead of "post_messages". There is also an error message for when an unknown request is given.

The client.sh script will interact with the server.sh script by representing a client running a command, with that client being the $id used as a parameter. This script is similar to server.sh in that once run it begins an infinite loop, where the user can repeatedly enter commands. The commands are sent to server.sh, and from there server.sh runs the script associated with that command.

4. The named pipes we used were server_pipe and "$id"_pipe, with the latter being a pipe named after a user. The clients will write all of their requests to the server pipe, while the server pipe will reply using individual pipes named after the client being replied to, to ensure security. The names chosen were simplistic; they are named after the script that will read from them.

5. Locking was performed by two scripts acquire.sh and release.sh, which as their names suggest, applied or released locks on scripts. At certain points in the scripts create.sh, add_friend.sh and post_messages.sh, acquire.sh will be called, meaning that no other terminal will access that script until it is released. The release.sh script will be placed at every instance where the script has provided an output, for instance in create.sh, regardless of whether no identifier was given, the user already existed, or the user being created, release.sh will be called to allow other clients to use the script.

6. One of the most consistently difficult aspects of this project was repeatedly retrofitting existing scripts with new features, which could lead to having to rewrite it from scratch. display_wall.sh, for instance, originally used cat to print out all its contents, which later had to be changed so that all its contents could fit in the pipe and not just the first or last line. In addition, it had to receive a new parameter, $requester, in the event of a user requesting to display a wall that wasn't theirs, as with just one parameter the wall contents would've gone to the owner's pipe, which isn't necessarily the requester.

7. One feature we debated on including but decided against was a scenario in which entering the command "display" in client.sh but not specifying a user would default it to the current $id, that being the one that called client.sh. This could be done by checking to see if the command was display but the parameter following it was null.

8. In conclusion, we successfully created an application that mirrors Facebook, where users can become friends with each other and comment on each other's walls, with the help of named pipes and locking mechanisms.