CT213 Project: BashBook

Group Name: Kevin and The Rock
Ranya Nike Shirin Praxmarer, 22314361
Ashlyn Buckley, 22440672

Introduction:
This project's aim is to create a rudimentary version of a Facebook-like social media platform that we run on the terminal. It allows you to create a "user", add another "user", post a message to someone's "wall" and display your own "wall". We will use bash scripting to create scripts that accomplish these operations as well as a client-side and server-side to send requests between. We use named pipes to simulate the requests being sent. We also incorporate a locking strategy in order to limit synchronisation errors and uphold the integrity of the operations. Throughout this project, we hope to hone our debugging skills as well as improve our overall understanding of Bash.

Implemented features:
- Create
- Add
- Post message
- Display wall
- Server
- Client
- Locking strategy
- Named pipes

NOTE: We coded everything sitting together in a pair programming style. We would begin each script by writing out pseudocode about how we think the code should work logically then would begin writing bash. We would solve errors together, both researching the problem with online resources. Ranya would type the code out in her VM as we solved each part, then would upload the scripts to our shared repository on GitHub so Ashlyn could go in and type up the comments.

Description of the named pipes:
There is one server pipe created in server.sh. All queries from clients are sent through there to the server.
It reads whatever is currently in the server pipe and assigns it to a temporary variable, we then funnel the values in the temporary variable into an array. Then these values are spliced based on a delimiter into several variables, e.g. "create", "Charlie".
The first variable, which will always be the command, which is tested in a case statement. Upon finding the corresponding command, all variables are sent to the respective bash script, i.e. "create" => ./create.sh

Then, each command execution bash script performs the operation. Then in server.sh, it holds this result (because it called it, like how functions work) and sends that down the client pipe. There is an individual pipe for every user because since we entered a user id, it knows which client we are talking about. The id is part of the pathway to their respective pipe.

We make the pathway a variable that dynamically changes every time a new client is run on the terminal i.e. "…/Pipes/$id/client". We will check if it already exists, if it does, there's no need to make another. The response in the pipe is then posted on client side. Operation completed, then asks for the next request.

Locking strategy:

We decided that locking the operations as they're called is the most effective way to prevent synchronisation issues. For e.g. what if we tried to make two users of the same name at the same exact time? Possibly they'd create two of the same, breaking the logic of the program, or possibly only make one entity and two people end up sharing a friends list and wall. There are many ways this could go awry. This is how we identified the critical section of the code.
So, in server.sh, when we identify the correct command: We will acquire the lock for it, call the bash script, send the result down the pipe, and release the lock.

Challenges we faced:
We found part 1 of the project mostly straightforward. No issue took a long time to debug. Most of our issues came from the trial and error of learning a new language, i.e. Bash.

However, in part 2, we initially struggled to understand the implementation of named pipes. It took us a while to wrap our heads around the nature of them and how they actually work.

However, our main issue was that when we sent a request through client-side, nothing would happen. As we combed out small errors, we eventually realised the issue was in the actual command bash scripts. We were using exits wrong, we thought that for e.g., using it in an if statement, would just exit that statement – not the whole script. Interestingly, these scripts worked fine when we submitted them for review in part 1. Which meant it took a painfully long amount of time until we found the root of the issue. Once we understood the error and fixed everything the code ran perfectly.

A list of features we had not implemented but would've if we'd have more time:

- We thought about adding the ability to make your friends viewable through a friends list. We would've had to make it so it did not post your name as well (we set it up like that so you could post on your own wall).
- We also had the idea of allowing users to view other peoples' walls. This would've needed 2 arguments instead of one, the user who wants to view it, and the person whose wall it is.
- We would've implemented friends' permissions also, possibly having a separate txt file that the client added names to, i.e. "Charlie has permission to post on my wall". Then we'd check for Charlie's name, see it was there, then allow the post to be printed to the wall.txt.

Conclusion:
We successfully built a Facebook-like command-line social media network through using bash scripting. The main functions being creating users, adding users, posting messages to walls and display users' walls. We also successfully implemented a locking strategy and named pipes that would transfer command-line arguments for us.
Through this project, we have gained a deeper understanding of the abilities of the terminal as well as learning the Bash script language.