CT213 Computer Systems

BashBook Assignment

Students: Jake O’Grady (22464536)

Eon Griffin (22331331)

# Introduction:

This BashBook assignment aims to create a server using Ubuntu, a version of Linux that will replicate some of the actions of the social media site Facebook. To do this, our program must carry out the following actions. First, we will prompt the program's operator to create one or many users without duplication or blank name entries. Any users that get created will be able to post messages on other friend's walls and display the message walls of others. Finally, users can add friends similar to how you do so using Facebook. This means that we will have four scripts to carry out all of these requests and another two scripts for the client and principal of the program, making a total of six scripts to have this program operate.

One of the main difficulties we may encounter is if the server doesn’t reference a script correctly because the script is stored in the wrong location or the server is given an incorrect file path. Another problem we can foresee in this program is that the server has problems receiving the input that the user enters. But we are hopeful that we will be able to overcome these with some essential code solutions.

# Implemented Features:

There are four implemented features for this server. They consist of creating users, posting messages from one user account to another friend's message wall, Displaying other friends' pages(walls in this project) and adding friends. These implementations have been broken into different scripts and are described below.

## Create Users:

To create users in this program, we created a script named create.sh. This script is one of the more straightforward scripts in the project. This is because we only have to check if the user has inputted a name, and if they have, we just check the username with any names stored in the BashBook Directory. Once it's been verified that the name is valid and isn’t already used, a directory is created for the user, holding wall.txt and friends.txt files.

## Post Message:

The post\_messages.sh script manages the number of messages and the friend receiving the message. This script is responsible for ensuring that the friend the user sends the message to exists in the directory of friends. A successful message will be displayed to the user if the friend is found in the directory of the user’s friend.txt. The message will be formatted as ‘<sender>: <message>\n’.

## Display Wall:

This display\_wall.sh is responsible for displaying any posts on friend’s walls. The user first will call the friend whose wall they want to view. The script will then check if a user ID has been provided; if not, it will provide an error message. The script will do the same by checking an ID that has been entered against the IDs in the directory, and if it doesn’t match any, it will provide a user not found error. If the user ID entered passes the two checks above, the script will call the file wall file associated with the user ID in the directory.

## Add Friend:

The add\_friend.sh script is responsible for adding friends to the user. It first checks that the user you are accessing exists in the directory, and if it doesn’t, it will display an error message. It then checks the user with whom you want to become friends to ensure they are on the server. If the user you want to become friends with doesn’t exist on the server, an error message will be presented to you; otherwise, it will try to make you friends with the user if you aren’t already. If you are already friends with them, it will let you know. Otherwise, it will tell you that the friend has been added. The friendship process is bidirectional i.e. you will become friends with the user and the user will become friends with you.

## Server:

The server.sh script calls this project's other programs using an indefinite while loop. It is responsible for taking user task inputs and running the code associated with the switch statements if a valid one is entered. The server is also responsible for filtering the outputs to make them more user-friendly; for example, if the program outputs “nok: okay”, the server changes it to “Error user already exists”. We did this by taking the output from each program and removing the unecessary parts e.g using sed. We also used some RegExp to make text editing of output more efficient and clean. They will remove start and end tags from the program's walls before displaying the text for the user. The server will close the program if no predetermined command line is entered.

## Client:

The client.sh script first contains the bonus challenge, but the central role of the client is to take the input from the user command lines and output instructions for whatever command the user wants to execute. The client is also responsible for when the pipes are on and off. The script contains a sleep1 command, which delays the program from running for one second to delay synchronisation. We also inserted two trap functions to run parts of the script which will delete the user pipes when the user client closes. When certain signals (e.g ctrl+c or exiting the terminal) are trapped these functions will run and remove the user pipes and the mylock (responsible for the locking function).

Pipes within this Project:

A central server processes each user's requests using pipes in this project. For this particular system, all of the client's or users' instructions will be inputted to the server through one pipe, known as a server-named pipe. To keep the information outputted to every user private, we gave each user their pipe for the server output to be outputted more securely and privately. The user sends their input to the server using the server pipe. Due to this synchronisation issues may occur.

The user defines each of the private pipes that deliver the server outputs.

# Locking Strategy:

For our project, we decided to use the locking strategy Spinlock. We found that this locking strategy worked well for our project. However, as we progressed with the project, we figured that this strategy works well for small projects but would need to be more efficient for large-scale projects. One of the main reasons it wouldn’t be suitable for large-scale projects is that it tends to use lots of CPU and that users have to queue to carry out any commands, leading to long wait times if this is large-scale. If we were to scale up this project, we would probably have to consider changing to a more robust locking strategy. One reason this locking strategy isn’t used as often is that there is no deadlock prevention.

# Challenges:

The most significant problem in this assignment was trying to get the locks to work in the project. If we were to revisit this project or had more time, we would investigate and most likely select a different lock for this project.

# Features Not Implemented:

Overall, we didn’t leave out any features the brief asked us to. However, if we had more time to complete this assignment, we would have conducted more research on the locking system and looked for solutions to deadlocking, as we don’t want to happen in our program. Even the spinlock implemented could have been made more efficient we feel.

A list of implemented features:

Jake O Grady worked on create.sh and display\_wall.sh.

Eoin worked on post\_messages.sh and add\_friends.sh.

The server and client scripts were a joint collaboration. Jake worked on the pipe and locking mechanisms while Eoin worked on the while loops and switch statements integrating the previous Assignment part’s scripts.

# Conclusion:

In Conclusion, we are happy with the end product of our work. The most challenging part of this project was getting the locking strategy implemented, but other than that, it went pretty smoothly. If we were to do it again, we would have managed our time better as we didn’t allocate enough to investigate the locking system entirely, which we were going to use, and this part turned out to be the most time-consuming. After completing this project, we better understand using, communicating and running bash scripts.