

Ciaran Gray 22427722
Oisin McLaughlin 22441106

Intro:

This project consists of using bash scripts to create a social media-like program called bashbook, which allows users to log in and add friends (other users), post messages to friend's walls, and display their own message walls. We created the following scripts; server.sh, which is the server of the program which takes the command the user requests to perform and calls the other scripts to perform these commands, create.sh, which creates a directory for the user and checks for arguments and if the user already exists, add_friend.sh, which checks for the arguments user and friend, and then adds the friend's id to the user's friend txt file after checking if both the user and friend's directories exist, also we choose to implement the friendships as asymmetrical, so as that adding someone as a friend doesn't imply that you are their friend, post_message, which checks for the arguments of sender receiver and message, and then after checking if the sender and receiver's directories exist, and that the sender is in the receiver's friend list txt file, will post the message onto the receiver's wall txt file, display_wall.sh, which checks the argument for the user, and then concatenates the wall txt file to the terminal screen to allow the user to see the messages on their wall, and finally client.sh, which is run to connect the user to the server and uses named pipes, server.pipe, to send arguments to the server, to which the server sends back the output to the client through the user.pipe, which is unique for every user's id, to ensure each user receives the output for only their requests to the server. This project also required a synchronization lock, to ensure that each user process could only be accessed 1 at a time, but we were unable to implement this feature at this time.

Implemented features:

Ciaran.

Named pipe creation of server.pipe and user.pipe in client.sh using the mkfifo command

Process requesting in server.sh by reading in requests from the server pipe and creating an array to store the request so as that different processes can highlight a certain input by accessing an index of the array, for example, userID will always be at index 1.

Command switch for request handling in server.sh which checks if the request contains create, add, post or display and processes the request appropriately.

User directory and files creation in create.sh using the mkdir and touch commands

Checking user existence in create, add_friend, post_message and display_wall.sh using if statements

For adding friends, we added the name of the userID to the friend's txt file using echo command

The same process applies for adding a message to someone's wall

For checking if a user is a friend of another user, the grep -q command is used to check if the user's name appears in the friend's txt file

For displaying a user's wall, the cat command is used to concatenate the file to the terminal screen

Oisin

Using named pipes to send requests to the server through the server pipe and echo command

Reading in output into client.sh using the user's own named pipe user.pipe

Reading the response from the server pipe and storing the request into the array in server.sh

Pipe cleaning, in client.sh the server pipe is cleaned after every iteration of the loop

Trap for pipe cleanup on exit in client.sh

Named pipes.

Unfortunately, it seems that part of the problem with our project is due to our poor use of named pipes, it seems that our pipes aren't updating frequently enough, and some old requests get stuck in the pipes and ruin the flow of our program. However, we attempted to implement the named pipes feature to allow multiple user's to access the server at the same time by creating 2 named pipes, server.pipe, which sends requests from the client to the server, and "user".pipe, which is unique for every user and displays the output from the server to the client script. The reason the pipes are unique for each user is to avoid users receiving the output for other user's requests.

Challenges.

Most of our challenges came with the usage of our named pipes. We were unable to clear the pipes from containing old requests, and this created problems with our create, add, post and display scripts receiving the incorrect parameters. Another challenge faced was our server loops not running infinitely, and sometimes it would stop after the user enters one input. A challenge we faced and overcame however, was for part 1 of the project when sending in our arguments into the server script, we had problems storing the message when it contained spaces, and the program would only accept the first word. We overcame this by storing the arguments in an array, and this meant that if the message had spaces we could take the words stored from the third index upwards, and could store the entire message.

Features we have not implemented:

We have not implemented the lock feature as we had a fair share of problems with the named pipes already, if we were to add in a lock feature we would probably do something along the lines of creating 2 more scripts, aquire.sh and lock.sh, which would take in the name of the script we are trying to use. Create.sh would be added to aquire when it is called from the create script, and it would be removed from the lock once release.sh is called from the create script. This means that if another user tries to call create.sh, it would try to aquire the lock but since the lock has already been aquired by reate.sh for another user, it will have to wait using a sleep command and loop that continuously checks for the lock to be, and wait until the process that aquired the lock completes it's request and releases the lock.

Conclusion.

In conclusion, our project works as described for part 1. We can implement the social media aspect in bash. However, our script isn't quite ready for multiple users to access the server at the same time just yet as we experienced set backs while trying to implement the named pipes and haven't yet implemented the locking system.