CT213 Assignment 2

Michael Mc Curtin, Tomasz Gruca

Introduction

BashBook 2 is an updated version of the BashBook project that allows multiple users to connect to a single BashBook server simultaneously.

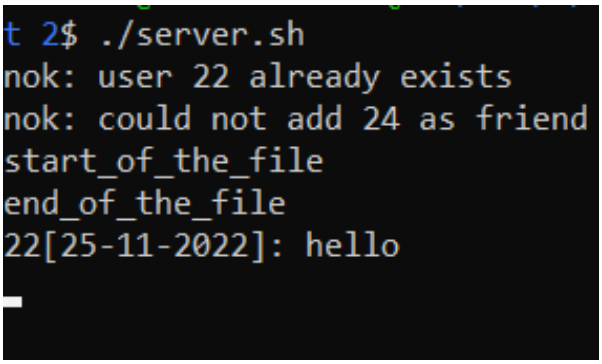
System Organisation

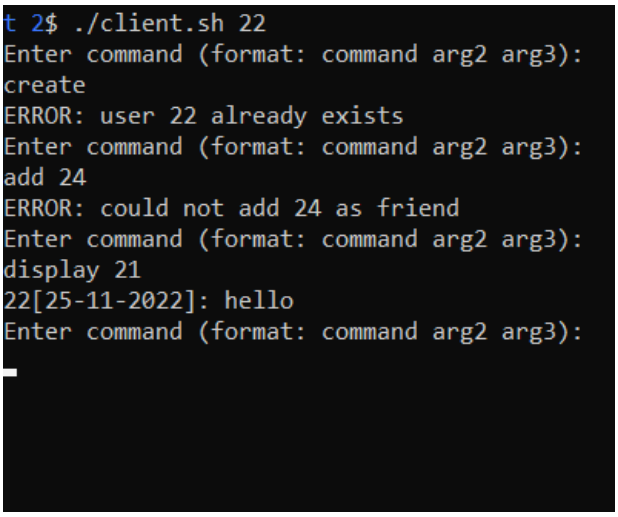
The social network revolves around four main functions, each with its own script:

1. Creating users (**create.sh**)
2. Adding users to a user's friends list (**add_friend.sh**)
3. Posting messages on a user's wall (**post_messages.sh**)
4. Displaying a user's wall (**display_wall.sh**)

These commands are all triggered by the server script **server.sh**, which runs constantly.

New in BashBook 2 is **client.sh**, a script through which the end-user inputs commands to the server. Client.sh is tied to the user's ID and communicates with the server via named pipes to ensure privacy.





Sample output of the BashBook 2 command line

Implementation

Tomasz and I divided responsibility between the new client.sh script and the modifications to the existing scripts. Tomasz wrote the client.sh script and I handled the modifications, including the implementation of the locking strategy.

create.sh

Creates a directory for each user containing two files: friends.txt and wall.txt.

If the input is invalid or the user already exists, no action will be performed.

add_friend.sh

Adds a friend to another user's friends.txt.

If either user does not exist or the users are already friends, no action will be performed.

post_messages.sh

Adds a specified message from one user to another user's wall.txt.

If either user does not exist or both users are not friends, no action will be performed.

display_wall.sh

Prints out a user's wall, line by line.

If the user does not exist, no action will be performed.

The output of all the above scripts is redirected to a temporary text file called serverout.txt. This is done so client.sh can modify the output to be more user-friendly.

server.sh

Creates a named pipe, *serverpipe*, for server input. Also creates serverout.txt

Runs an infinite loop which checks serverpipe for input.

This input is then checked for a valid request, if one is found then serverout.txt will be cleared, the desired script will be executed, serverout.txt will be written to *userpipe* and serverout.txt will be cleared again.

client.sh

Creates a named pipe, *userpipe*, based on the client's ID. Client.sh's arguments then are sent to *serverpipe*. Client.sh finally reads any output from serverout.txt, modifies it to be more user-friendly, then prints it.

Both userpipe and serverpipe are deleted upon exiting client.sh and server.sh.

Locking Strategy

Tomasz and I designed a locking mechanism based around the ln system call, which is supposed to be atomic on a local filesystem.

Each lock, named after the userID that it pertains to, is acquired and released via the acquire.sh and release.sh scripts respectively. The operations which write to files (namely add.sh and post.sh) would take and release this lock as they are executed in server.sh.

Unfortunately, upon implementing this lock I found that it froze server.sh. I could not find a satisfactory resolution to this issue and will take responsibility for its omission. However, in our testing we found that the commands were executed so quickly that we could not cause any synchronisation issues to occur. As BashBook's userbase grows, these problems will likely occur so this issue will need to be dealt with at some point in the future.

Challenges Faced

As I mentioned above, I could not find a satisfactory resolution to the locking strategy causing a system freeze and unfortunately had to omit it from the final product.

Working with named pipes in Bash proved quite difficult for both of us as unlike a standard text file it was not as easy view the contents of each pipe outside of the script execution.

Before we decided to remove each pipe on script exit and clear serverout.txt frequently, we found that commands would build up in each file and cause commands not to execute. This was resolved when we made each files temporary to the script execution.

Conclusion

Overall, this was a challenging assignment, but we feel that the features and user experience of BashBook have been greatly enhanced, even if we could not achieve our full desired implementation.