FlopatBook Submission

Jay Popat 22346566 Vlad Florea 22409144

The project we are discussing is a social media application similar to Facebook - Flopatbook (naming inspired by our surnames), that provides a platform for users to interact, share messages, add friends, and create profiles. This project is a unique blend of technical skills and creativity, combining the power of Bash scripting with the principles of social media interaction.

FlopatBook Submission

Jay Popat 22346566 Vlad Florea 22409144

Implemented Features

1. **User-User Interaction**

   → **Add friend**

   - Facilitates the addition of a friend P2 to a user P1's friend list and vice versa. The operation is performed by appending the P2's ID to P1's friend list file. This gives P2 access to P1's wall file. P2 will still have to add P1 to his own friend list to allow P1 to view his wall file.

   → **Create user**

   - Creates a directory in the user's folder with the name of user and 2 text files in the user's directory – "friendList.txt" and "wallFile.txt"

   → **Post Messages**

   - User adds a message into the receivers "wallFile.txt" -- The script appends the text into the file

   → **Display Wall**

   - Displays the wall file (chats) of the user passed in as a parameter.

2. **Server-Side Social Interaction**

   - This refers to handling social interactions on the server side. The user request and arguments are read in from the server.pipe and if they pass the error handling logic, the scripts are called and the SUCCESS/ERROR message is added to the user.pipe, ready to be outputted from the client script

3. **User-Server Communication**

   - The user sends across the request through the client script when prompted for a request and arguments. This data is sent into a server.pipe which is interpreted and further processed (including error-handling) in the server. The user is also passed in as an argument-- Which is used in the social interaction scripts which are called by the server.

4. **User-Server Communication**

   - The ERROR/SUCCESS messages are outputted from the server into the corrersponding user.pipe which is then outputted to the screen client-side. On the other hand, the server.pipe is used to send user request data to the server.

FlopatBook Submission

Jay Popat 22346566 Vlad Florea 22409144

Description of the Named Pipes and their Naming

Server Pipe - The server pipe, named "server.pipe", is designed for readability and ease of use. This pipe serves as a conduit for all client requests, directing them towards the server for processing, hence the name. The server reads from this pipe, interpreting/processing the requests and executing the corresponding scripts to fulfill the user's request. The client will write new requests in the form <request> <id> <arguments> to the server.pipe and this will be received and processed in the server.

User Pipe - The naming was done in the format "user".pipe so that each user has a distinct pipe for themselves. This will be used to send the output of the server back to the client. The server will write the ERROR/SUCCESS messages to the pipe and this will be received and printed out client-side. The contents of the pipe are read by first storing them in a file "serverOutput" and then reading from that file (cat)-- The message/contents of the file is then displayed to the screen on the client.

Description of the Locking Strategy

The locking mechanism is implemented in the server itself. The idea behind it is that, once the server receives the request that came through the server.pipe, it would then enter the processing section defined by the switch statement. This is the critical section of the program. Thus, the server acquires the lock "mainLock.txt" and executes the critical section. The server handles the request and then sends the output back to the client through the corresponding "user".pipe. Once the request is handled, the loop will be back at the start, at which point it is no longer in the critical section and the lock is released.

Let's describe a scenario-- Two people called "Jake" wish to create an account. Without locks, Jake1 and Jake2 both enter a race condition once they reach the critical section in the server. A possible outcome is both Jakes pass the error handling successfully BUT Jake1 creates his user first ("jake") and then Jake2 tries to also create his user as he has passed the error checks HOWEVER the create user script fails as a "jake" user directory was already created by Jake1. This will cause unwanted errors in the program.

Now let's use the same scenario but implement the locks (implementation described above). Jake1 acquires the lock first. Jake2 then tries to acquire the lock but Jake1 already has it. Jake1 then creates his user "jake" and releases the lock. Jake2 then acquires the lock and tries to create his user "jake", however he must first pass through the error handling which will now prevent Jake2 from creating the user "jake" as that user already exists.

FlopatBook Submission

Jay Popat 22346566 Vlad Florea 22409144

Challenges faced

Posting messages - If the message we posted was separated by whitespaces - it was counted as different arguments which led to problems. We had to research and hence design an implementation which concatenated these into one argument.

Pipe functionality – It took us a while to figure out the client-server connection using pipes. We had to experiment and do more research so that we could visualise how to connect the two entities together.

Features we could implement if given more time

1. **User Authentication and Authorization**: This feature would handle user login, registration, and permissions. It could use Bash scripting to interact with a user database and validate user credentials. It could also check if a user is already logged in to not have multiple instances of the same user.
2. **Message Encryption**: To ensure the privacy of user messages, we could add a feature that encrypts messages before they are sent and decrypts them when they are received. This could be done using Bash scripting in combination with encryption tools like OpenSSL.
3. **User Notifications**: This feature would notify users of new messages posted to their wall and friends added etc. Bash scripting could be used to interact with the notification system and deliver notifications to users.

Conclusion

In conclusion, this project has provided an insightful exploration into the development of bash scripts and the interaction between entities using pipes, locks etc. This provided us with a greater understanding of how processes communicate with each other using pipes and how mutual exclusion is achieved through the use of locks in Bash. We are both on Linux (we use arch btw) for the past year and have learned Linux concepts and Bash scripting in our spare time (arch user problems y'know). This project made us more motivated to program more command line applications and write scripts to automate our daily workflow on our ***ARCH*** (btw) machines.