Project Report: Social Media System

Intro/Requirements

I used bash scripts to create a social media system where clients can send requests to a server to create a user (which has a friends list and a message wall), add another user to a user's friends list, post a message on the wall of another user, show a user's wall (display their messages) and shutdown the server (though in a real system the clients shouldn't be able to shutdown the server for everyone).

To implement these requests, I wrote a script for each request, create.sh, add.sh, post.sh and show.sh, a server.sh script to process each request and a client.sh script to send requests to the server.

Architecture

The system works by clients sending their requests to the server and receiving the reply using named pipes.

The create.sh script checks that it received one command-line argument, $user, and creates a directory with that name. The directory contains two files, friends (which contains all users added as friends) and wall (which contains all the messages posted to the user).

The add.sh script checks that it received two arguments, $user and $friend, checks that each argument is the name of a directory, checks that the $friend isn't already in the friends file (using grep) and then adds the $friend to the friends file.

The post.sh script checks that it received 3 arguments, $receiver, $sender and $message, checks that the $receiver and $sender are directories, checks that the $sender is in the $receiver's friends file, and then adds the $message to the $receiver's wall file.

The show.sh script checks that it received one argument, $user, checks if the $user is a directory, and then displays the $users wall file.

The server.sh script creates a named pipe, server.pipe, and then uses an infinite loop to constantly check the server.pipe for messages from the client.sh script. It reads the message from the server.pipe into 5 variables, id, request and the request arguments. It uses a case statement to check the second argument, the request, and then lock the file that will be used in the request, execute the request using the request script, the output of this script is sent to the clientId.pipe, and then unlock the file. The script deletes the server.pipe and exits when a shutdown request or an invalid request is sent.

The clients.sh script checks that it received at least two arguments, a clientId, a request and the request arguments (if needed) as command-line arguments. It creates a named pipe, clientId.pipe, to receive the reply from the server and then sends the request to the server.pipe. It uses a case statement to check that the number of arguments provided is correct, and send the correct message to the server.pipe for each request (e.g. sending 3 arguments for the create request, clientId, create, $user or 5 arguments for the post request, clientId, post, $receiver, $sender, $message). It then reads the message from the clientId.pipe, and then deletes the clientIdpipe. If a shutdown or invalid request is entered, the script deletes the clientId.pipe and exits.

Conclusion

I didn't really face any challenges in writing the scripts, I thought it was pretty straightforward as I knew from the labs how to use pipes and the if, while and case statements. The only problem I had was locking, I'm not sure if what I did is sufficient, using the lockfile command in the server.sh script to lock the user's friends or wall file for the add, post and show requests. I wasn't sure where else you would need to lock the files.

I enjoyed writing the scripts, as it was different to the other modules and I found it interesting to see how a server works.