# Social Media System

**Introduction:**

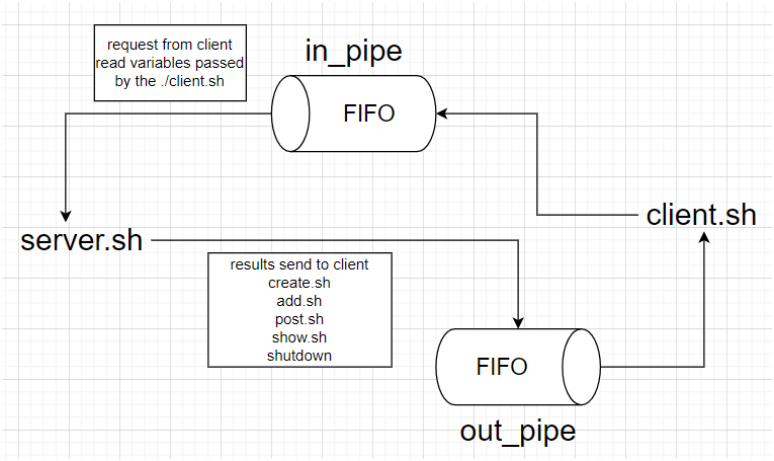
This project enlightens about the functioning of a social media application. Implementing such a system gave more exposure to Operating Systems concepts and power of scripting commands. It indulges concepts such as use of variables, inter process communication, error handling etc. It covers the knowledge of bash conditions, arguments passing string slicing and semaphores. This project is strongly emphasizing on server client communication where request like create, add, post, show drive the execution of pieces of scripts on server side.

**Requirement:**

The initial requirement for the system is that there must include a server and a client to interact with one another. This being a social media system one must expect a userID, friends, wall and comments. This system basically allows a user to create their id's account also make friends and post on their wall. Keeping all in mind also invalid entries should not be made. Thus, error should be thrown and handled accordingly. Exit cases must be included for such cases. Both the server as well client create a named pipe at their end for communication. Wherever the users might access a script at the same time it could end up having a concurrency state. To avoid that a lock could be implemented so that it can appended or accessed simultaneously. The client script must every time check the given list of parameters to the script and an id passed to the server side with correct arguments. If the arguments passed do not match the input condition the case statement will not work and show an error message.

**System architecture:**

The diagram below shows the functioning of this system




Starting with the create script, this will first check if entered arguments are equal to 1 or not. If more or less than 1 throw error message: "parameters problem". Otherwise create a directory for the user and build files named friends and wall within user's folder. Also, if user tries to recreate folder deny permission with message "user already exists".

```
cs20201672@csserver:~/project-rohitdalvi96$ ./create.sh
Error: parameters problem
cs20201672@csserver:~/project-rohitdalvi96$ ./create.sh anthony
Error: user already exists
cs20201672@csserver:~/project-rohitdalvi96$ ./create.sh paul
OK: user created
cs20201672@csserver:~/project-rohitdalvi96$
```

Then adding a friend feature lets user make another existing user their friend. No one without its directory present in the system can make friends. This script checks for 2 parameters if not pass an error to the user. This script goes through series of existence checks where it searches for an existing folder. This makes sure whether both users exist in the system or not. Then we see if the user is already friend of another user or not by searching the variable value passed as string using grep. This will peek into the friends file and check for the username. If no matching string found it adds user as friend.

```
cs20201672@csserver:~/project-rohitdalvi96$ ./add.sh paul
Error: parameters problem
cs20201672@csserver:~/project-rohitdalvi96$ ./add.sh anthony tony
Error: friend does not exist
cs20201672@csserver:~/project-rohitdalvi96$ ./add.sh tony anthony
Error: user does not exist
cs20201672@csserver:~/project-rohitdalvi96$ ./add.sh anthony thomas
Error: user already friends with this user
```

After adding two members who are friends, they can post on each other's walls. To do this only friend can post to a user's wall. Another case when user1 is not friend of user3, user1 cannot post to user3's wall. This script can accept only 3 parameters which is both the usernames and a message to be posted.

```
cs20201672@csserver:~/project-rohitdalvi96$ ./post.sh anthony
Error: parameters problem
cs20201672@csserver:~/project-rohitdalvi96$ ./post.sh anthony tony "Hey there"
Error: Sender does not exist
cs20201672@csserver:~/project-rohitdalvi96$ ./post.sh tony anthony "Hey there
> "
Error: Receiver does not exist
cs20201672@csserver:~/project-rohitdalvi96$ ./post.sh thomas anthony "Hey there"
Ok: Message posted to wall
```

show script allows to view the wall of the user where all the posted messages are printed. These messages are displayed between wall start and wall end texts. It requires only one parameter like the user's wall to be viewed. Using the cat command, we can achieve this result.

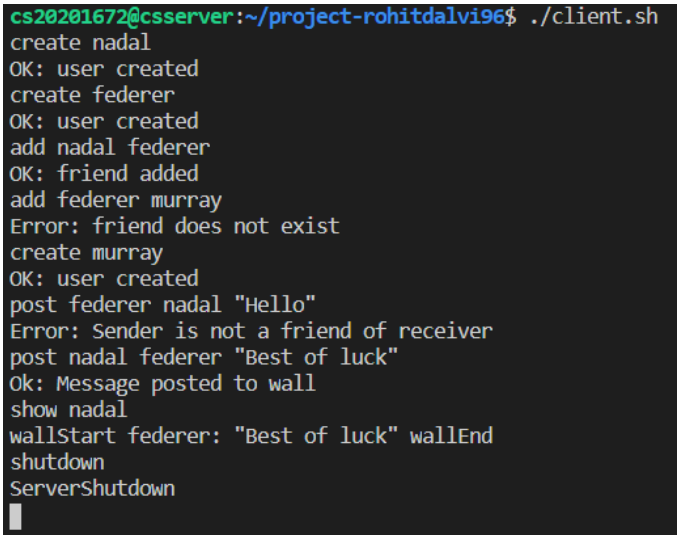
```
cs20201672@csserver:~/project-rohitdalvi96$ ./show.sh
Error: parameters problem
cs20201672@csserver:~/project-rohitdalvi96$ ./show.sh tony
Error: user does not exist
cs20201672@csserver:~/project-rohitdalvi96$ ./show.sh anthony
wallStart
thomas: hello
thomas: How are you
wallEnd
```

The server script is one of the drivers of the communication system. Its responsibility is to read request

made from the client store it in a variable and perform action accordingly. Now the transfer of messages is achieved with the means of named pipes. In my project I have created two named pipes in_pipe and out_pipe. These pipes work according to first in first out manner which means first entered operation must be exited from the pipe first and so on. At the beginning the server.sh looks for any present pipes in the directory. If it exists, then remove that pipe else create one using mkfifo $pipename. The trap command helps catch signal, in my case to delete the files on exit instead of deleting them at every required instance. Once while loop runs the requests are read by the input pipe and within the case statements one after the request scripts are placed to be executed as per demand from client. When server gets a request, it enters the case body and executes the .sh scripts and echo the output to outpipe. As defined earlier in every basic command scripts according to the number of parameters passed the server will execute these statements and provide result to the client as an input. While it is mentioned in the pdf that final output must be displayed at the client's side messages must be redirected to the requester that is client using out_pipe. At the end if case statements conditions aren't fulfilled then an error message will be shown.

```
cs20201672@csserver:~/project-rohitdalvi96$ ./servr.sh
create anthony
OK: user created
create anthony
Error: user already exists
create thomas
OK: user created
add anthony thomas
OK: friend added
post thomas anthony hello
Error: Sender is not a friend of receiver
post anthony thomas hello
Ok: Message posted to wall
post anthony thomas "hello there"
Ok: Message posted to wall
show thomas
wallStart
wallEnd
show anthony
wallStart
thomas: hello
thomas: "hello there"
wallEnd
shutdown
cs20201672@csserver:~/project-rohitdalvi96$
```

The client side of the application hold the responsibility of writing the request to the server. For the user to enter commands the loop must run continuously until it requests the server to shutdown which will eventually close all the further operations on the terminal. Once the ./client.sh script is run it prompts for input from user. After entering the request with arguments, it sends execution request to server and displays its result on client terminal window. The procedure is shown in the below snapshot.




With reference to the above snapshot a working solution to this problem has been displayed. The stepwise implementation of a social media network shows how a user gets created at first. Thus, after both the directories are formed then those can be added as friends. The third scenario where a new non-existing user tries to be added as friend. The request gets denied as that directory does not exists. Next when a person post on the wall it must a case where the other user must be a friend of the user who is posting on the wall. Otherwise, an error message gets displayed saying sender is not friend of receiver. For the error handling this script first checks if the input pipe is created or not because if the pipe won't exist the outpipe won't be able to communicate with server by any means. While each response flow through the pipe in a fifo manner there might occur an issue of concurrency. Thus to avoid the blockage in execution a minute sleep time is introduced between every transfer process. To print all the results on the client's terminal those outcomes are redirected to outpipe and echo. Finally, when all the execution is completed it releases exit 0 which stands for no error and breaks the loop.

## Overcoming challenges:

While initiating the project visualizing what is expected out of this idea and how to frame logical questions for understanding the approach took considerable time. But the strategy to complete the task stage by stage helped to know the ask of this project. Linux syntax being fresh to our knowledge consumed time, but the lectures slides, and tutorials were much handy when it came to writing logic for the basic commands portion. Navigation into directories using combination of commands was tricky at first. In the server.sh script I started by analyzing the closest method to take inputs from the user and find which shell script to be run. Exploring about executing a shell script into another gave me an option to use the read -p command which allows to read the variables all together. The -p stands for printing the prompt before read without new line. After reading executing them using the exec command with appropriate syntax to keep the server running continuously till its shutdown on request.

## Conclusion:

This project overall taught to identify the optimum utility of basic commands in Linux which helped in creating a social media model. There are various methods to fight against a problem which defines the complexity of error handling one can implement in their code. Shared resources and process consumption denotes the waiting time and deadlock condition if the process does not get its resources

on time. The concepts covered in class were used to the fullest which did minimize the challenges. Entire process of gathering requirement developing and deploying the code to GitHub was an industrial experience. Implementing realistic background to the model by starting and shutting down the server and letting the client know about its activity shows readiness of the system.