Operating system (conv)

COMP30640 Project 2019
Password Management System

Sachin Soman 19200494

Introduction

The OS project for 2019 involves the implementation of a shell-based password management system. Users will be able to store and retrieve passwords via a command-line interface (CLI), with front end made of a client that passes commands to a server to retrieve or edit information.

This document contains information regarding the architecture of the program, implementation, challenges faced, concepts used, possible bugs and additional implementations for the program.

Requirements

The program must have client-server type architecture. The user will be able to enter various commands into the client which passes the information to a server that executes it.

The main requirements are:

1. The program must be able to create new users
2. The newly created users can add services which can be removed or modified by the user
3. Each service is a file that contains the user login and password
4. The program must be able to retrieve the password at the request of the user
5. The program must be designed in such a way that multiple users can send a request to the server and not cause inconsistencies ie synchronization must be handled utilizing semaphores.
6. The program must use pipes for inter-process communication.

Architecture

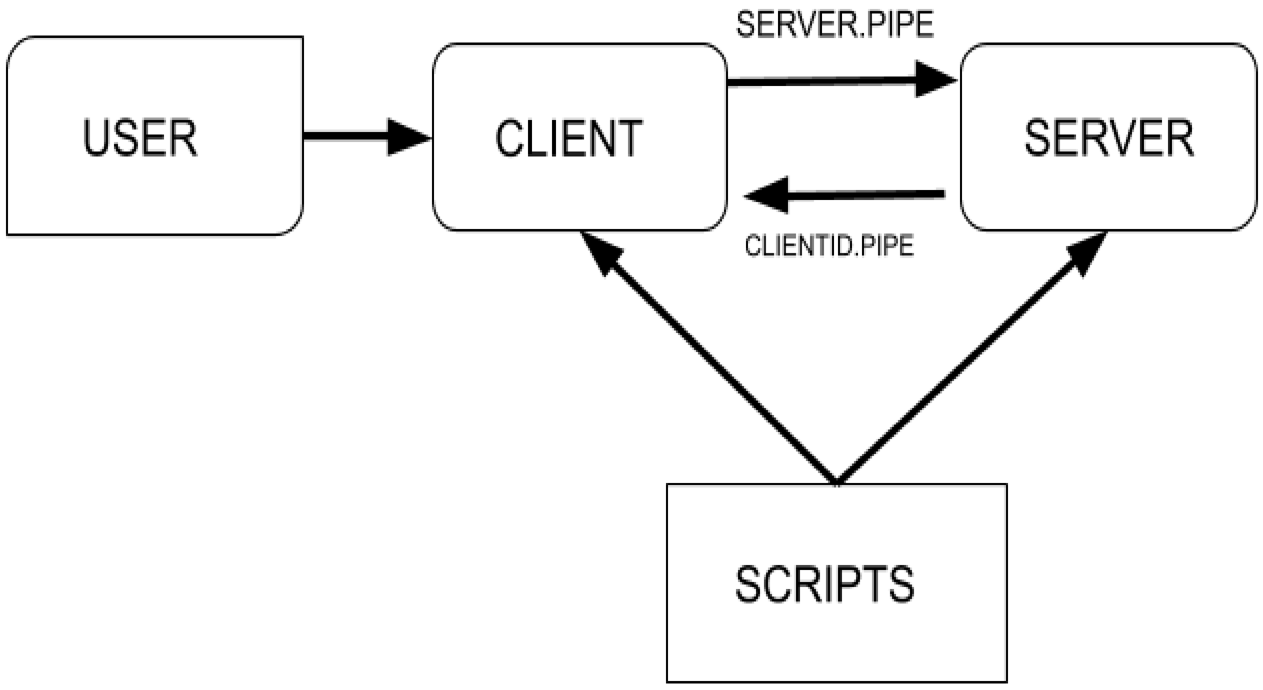
The password manager consists of 3 blocks.

1. Client - The user-facing interface where all the instructions are passed and messages are received.

2. Server - The backend which receives data from the client and calls the scripts

3. Scripts - The files which contain individual programs which make up the system

4. Pipes - Interprocess communication channels used to transfer data from Client to Server to scripts and back to the client. Here it must be mentioned that named pipes are used for IPC than normal bash pipes.



DESIGN CHALLENGES

During the development of the project, each new script brought new challenges, difficulties, and solutions. The next section is devoted to present all the observations

Scripts

- Init.sh

  The init.sh script is used to create a directory which represents a user; The script is designed to accept only one parameter which is the name of the user and must be passed in the client as

  ./init.sh $username

  This was the first script to be implemented, concepts such as argument passing and numbering of parameters were used. Learned how to use comparison operators and conditions in bash. The script was straight forward and no possible bugs or issues where observed. The script needed to be modified later to include semaphores to allow only one user to make a file at a time. At times it is observed that after making users concurrently the symbolic link used doesn't get removed. This doesn't occur most of the time to find out the reason, but when it does occur I have to manually remove the lock created.

- Insert.sh

  The insert.sh script is used to create services inside user directories; The script was initially designed to accept only 3 parameters the username, service and the payload.

  ./insert.sh $user $service $payload

  The main problem faced in developing the script was in parsing the arguments received based on '/' , for this internal field separator was used IFS = '/ '. This solves most of the issue however the trade-off was that username or payload cannot have '/' as a character. This observation came quite late when I did final testing by that time the entire program used IFS =' / '. In the future, a more robust system by use of SED is planned to be employed.

- Show.sh

  In show.sh script the `cat` command was used to display contents from the file the main challenge in this was finding out all different combinations in which parameters for user and service of multiple depths can be passed.

  There might be a chance that I might not have considered a combination of user service combination for parsing parameter however during personal testing there were no issues

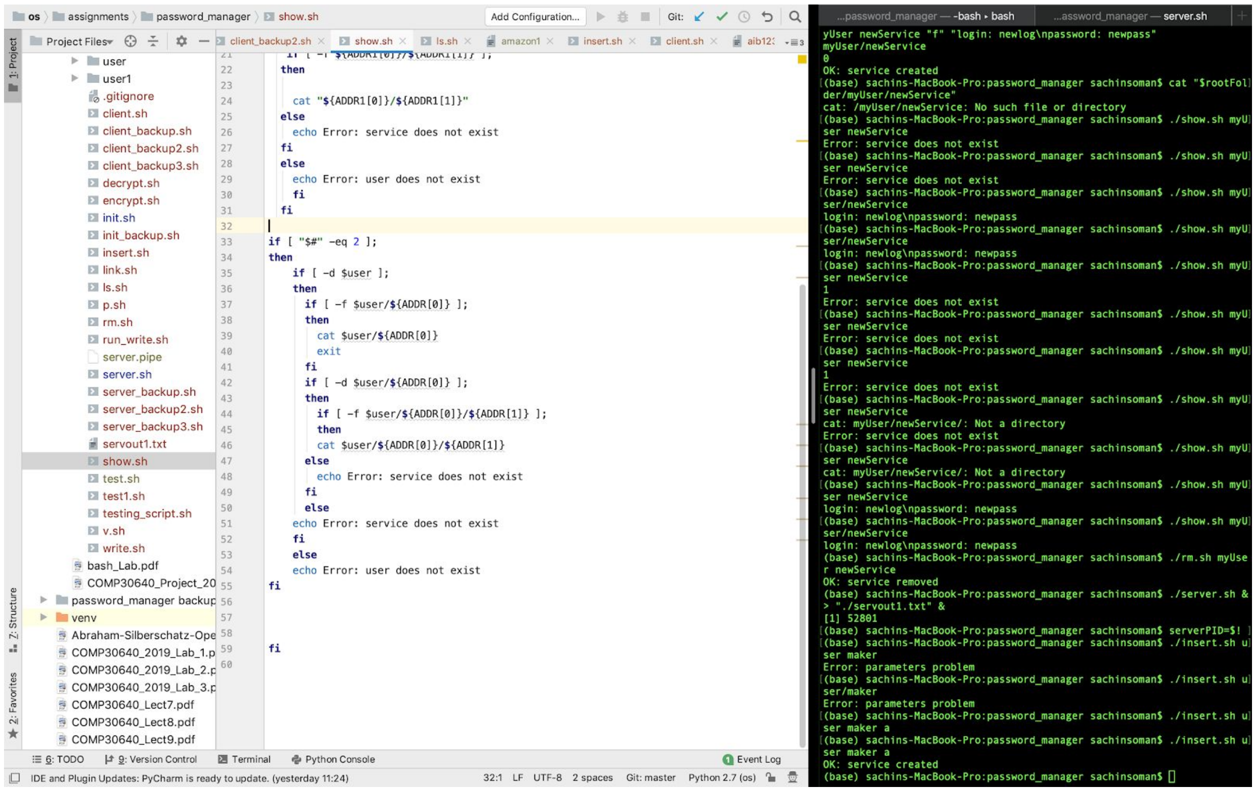
<u>NOTE</u>

For show to work properly the service must be first created via client as I have a certain format in which file must be written for show to work properly.

./show.sh $user $service

<u>NOTE</u>

A major issue during the development was the frequent disconnection of the ssh connection therefor most of the development was done locally using pycharm ide with mac default terminal for running the scripts. This setup significantly improved the efficiency of writing the scripts and testing it. The only issue that was encountered was the lack of tree command in the mac terminal which was later installed using homebrew.



```
    if [ -f "${ADDR1[0]}/${ADDR1[1]}" ];
    then

      cat "${ADDR1[0]}/${ADDR1[1]}"
    else
      echo Error: service does not exist
    fi
    else
      echo Error: user does not exist
      fi
    fi

if [ "$#" -eq 2 ];
then
    if [ -d $user ];
    then
      if [ -f $user/${ADDR[0]} ];
      then
        cat $user/${ADDR[0]}
        exit
      fi
      if [ -d $user/${ADDR[0]} ];
      then
        if [ -f $user/${ADDR[0]}/${ADDR[1]} ];
        then
        cat $user/${ADDR[0]}/${ADDR[1]}
      else
        echo Error: service does not exist
      fi
      else
    echo Error: service does not exist
    fi
    else
    echo Error: user does not exist
fi



fi
```

```
yUser newService "f" "login: newlog\npassword: newpass"
myUser/newService
0
OK: service created
(base) sachins-MacBook-Pro:password_manager sachinsoman$ cat "$rootFol
der/myUser/newService"
cat: /myUser/newService: No such file or directory
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser/newService
login: newlog\npassword: newpass
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser/newService
login: newlog\npassword: newpass
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
1
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
1
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
cat: myUser/newService/: Not a directory
Error: service does not exist
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
cat: myUser/newService/: Not a directory
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser newService
login: newlog\npassword: newpass
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./show.sh myU
ser/newService
login: newlog\npassword: newpass
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./rm.sh myUse
r newService
OK: service removed
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./server.sh &
> "./servout1.txt" &
[1] 52801
(base) sachins-MacBook-Pro:password_manager sachinsoman$ serverPID=$!
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./insert.sh u
ser maker
Error: parameters problem
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./insert.sh u
ser/maker
Error: parameters problem
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./insert.sh u
ser maker a
(base) sachins-MacBook-Pro:password_manager sachinsoman$ ./insert.sh u
ser maker a
OK: service created
(base) sachins-MacBook-Pro:password_manager sachinsoman$ 
```

- UPDATE SERVICE USING insert.sh
  The insert.sh script needs to be modified to accommodate a fourth parameter 'f' which if added will edit already existing service and save it. It is my interpretation from the question that either the insert command must have 3 parameters if so it will behave like initially designed insert script else if ' f ' is added it will edit and save the already existing script.
-

<u>NOTE</u>

If the normal behavior is to be expected only input 3 parameters instead of four don't add " " as the fourth parameter.

./insert.sh user1 Bank/aib.ie f "$payload

Or

./insert.sh $user $service $payload

- Rm.sh

  This script is used to remove the services , the rm command was used for the removal

  ./rm.sh user service

- Ls.sh

  The list command uses the tree command to list all the services which the user-created an interesting observation was the fact that the tree command was not available in mac terminal and therefore had to be installed using homebrew hence I was able to learn homebrew for installation of UNIX packages without worrying about dependencies.
- P.sh and V.sh

  These are the two files used to maintain semaphore the p.sh is from the modified script from os practical 5. At times I have used p and v to lock sections beyond the critical section. The one drawback to this practice is that in case the script needs to be stopped in between the lock file is not removed and a deadlock situation might arise.

- SERVER SCRIPT

  The server script consists of an infinitely running while loop with a series of switch cases. The server listens for commands employing a pipe (server.pipe) which will be made at the beginning, if the pipe is not already present. The server will receive a message from the client-side via the server pipe and is parsed and the switch cases will run an appropriate script based on the received command. The message from server.pipe is parsed using IFS. The messages from the server-side to the client-side are passed via named pipes which are unique to the client id sent. In the server, I have put a small sleep between the server.pipe read this is to ensure that the server doesn't start reading before the client stops writing in the and sends and a close signal to server.pipe. The result was an infinite loop on the server-side. This issue was not reproducible on demand but used to occur frequently hence the small sleep statements, This seems to have stopped the problem. The same problem happened to one of my classmates in the insert part, adding a small delay between read-write operation removed the issue.

NOTE

At times after using the same server.pipe for a large number of times at some point the pipe starts to send command which is partially missing words. I was not able to find out the reason for it and the only solution I could find was to delete the server.pipe and start the server again. Another limitation that was observed in pipes was when I encrypted service files it was observed that long strings where getting cut, after reading some documents the conclusion I can draw is that buffer for pipe will stop at the point where it can no longer guarantee an atomic operation.
Source : https://unix.stackexchange.com/questions/11946/how-big-is-the-pipe-buffer

A possible fix that I have employed is to encrypt only the password so that the string doesn't become too big from the openssl encryption algorithm.

- CLIENT.SH
  The client has a similar structure to that of the server. However, it is not running constantly by means of a while loop. Whenever the client is invoked a unique named pipe is made from the client id which is received as a parameter. The clientid.pipe is used to receive information back from the server.

  The client also has some scripts executing in it ; the decrypt script will be invoked one the payload is received from server and password gets decrypted. The edit script is also an interesting section it calls the show command and receives the payload then gets passed to a temporary file using 'mktemp' command and vim is called on it. The user can now edit the file and the insert.sh script with 'f' parameter gets invoked so that the information gets updated.

  An observation while implementing client was effect of using echo and cat to display the contents from pipes it was observed that when cat was used to output the contents of pipe at times the client will continue to run and I had to manually stop the process for this reason most of time contents from pipe are stored in a variable and then displayed using the echo command.

- ADDITIONAL IMPLEMENTATION:
  Encryption was implemented using openssl algorithm which was provided a number of issues popped up during the implementation , the first issue was “ bad magic number” error this happens because the decrypt script requires a new line to be present in the encrypted password use of echo to write file removes the problem as it automatically adds a new line at the end. Another issue is the problem of outputting special characters into a file using echo. If an encrypted password says has a character ‘/ ’ while writing it to a file it is recognized as a space between character, hence on trying to decrypt an error arises. A possible solution is the use of command -e in echo however upon checking man page in mac bash terminal -e command seems to be missing while -e is present in the terminal in cs server. Therefore -e is used in the echo statement. I was not able to get a proper solution in a Mac environment

Random password was also made an option in the client script.To generate a random password insert command is invoked when the option to input password type “R” and a random password is generated, this will also be encrypted. Random password uses the time as a parameter.

password=$(date '+%s')

CONCLUSION

A successfully working program was made, however, I was not able to make an exhaustive list of tests hence the program is still prone to bugs arising when the script stops unexpectedly. The project helped me in understanding thoroughly the essentials of bash as of now I was successfully able to read and understand most of the concepts that were conveyed through the book “The Linux Command Line” by William E Shotts. The concept of synchronization and the use of semaphores became apparent. The importance of bash especially for web scraping and automation was also understood. Another interesting skill that I was able to acquire was the ability to use CLI to create folders, files and do other processes much faster than by means of GUI. The process of installing additional packages for Linux by means of package installers like Homebrew was an added bonus.

The project initially seemed quite difficult and impossible to make however this exercise helped in understanding the importance of using documentation which seems to be the most important resource while developing a computer science project. And bash seems to be a very valuable tool in the hands of a developer

I would in the feedback like to receive some inputs on how to proceed forward with the newly acquired skill set.