COMP30640 Project Password Management System

Introduction

All of the criteria have been completed to the best of my knowledge and I have attached 11 scripts which I feel fulfil the brief outlined. I'm not sure if it was done the most efficient way possible but I think my scripts are somewhat easy to follow. I have also completed the bonus section password encryption. Outlined below is the design of my system and also any challenges I faced along the way.

What is the system supposed to do?

It is a password management system allowing the user to create, update, view and remove files storing their passwords for various services. The idea is to run the server script in one terminal and the client in another. By typing commands into the client these are passed to the server and subsequently onto the individual scripts for the actions to be performed. The resulting message is then passed on to the client script to be read by the user.

Architecture

I designed the scripts to work independently so init, insert, show, ls and rm can all be ran individually and parameters passed on their own.

There as a few features below that were not explicitly in the design brief but if I had more time I would have liked to try implement:

- In the client.sh Insert ask for log in details before checking if it exits. This should check if the file exists first before asking for the user to input any information.
- If more time I'd like to put in more user instructions and error handling. I have minimal instruction messages on how the user should use the system.
- I would also like to have run more tests to check the robustness of the system.
- In terms of encrypt and decrypt I have the client id as the master password. This may not be the most secure thing in the real world but felt it was ok for this assignment.
- If I had more time I'd like to enact the recommendations from shell checking my scripts.
- Initially every script has a check for parameters but I made this redundant in the client script which may be a flaw Id change if I had more time. I'm only passing the required variables from the client to the server rather than all of them $@.

Design

Init) This script is designed to let a user create a directory to store their file passwords. I have an if statement to check the number of parameters. The else if checks if the user directory already exists and if both these are ok if creates a user directory and echos a message as such.

I’ve put the semaphore before the final else before creating the user directory. To me this was the critical section of the script so was the part that needed to be protected.

Insert & update) This script allows a user who has been created in the previous script to insert a password file and if necessary, make a new directory within there. If the third parameter passed is in f then the user wishes to update the file but if it’s any other character then the user is trying to insert. First if statement checks the number of parameters. Two else if’s check firstly if the user directory does not exist and prints an error message as such and the second else if checks if the file exists but the user did not enter F meaning the user is trying to insert a file that exists rather than update it. If the file already exists an error message will be displayed. The final else statement is where I place my semaphore as this is the critical section where directories and files are updated/created. The first if either makes a directory ie Bank etc if it does not already exist and then enters the last section of if and else where the first if checks if the file exists and the user typed f in which case the user wishes to update a file that already exists so the original file will be over written and a message saying file updated will pop up. Else the file is created and a message echoing this will be displayed.

show) The idea of this script is to allow a user check the login and password of a file that they already created in the previous scripts. First if statements checks if the correct number of parameters are being passed, else if checks if the user directory exists and if it doesn’t displays an error message as such. The next elif checks if the file exists in which the user wants to display the passwords from. If it doesn’t then an error message is output. Finally if the file exists then I place a semaphore around the information I’m grepping to lock the user. My logic here is that by locking the user then the file password can not be changed while I’m trying to display it. I use grep -i in this case also as the -i ignores case which is beneficial if a user has their login store as Login etc.

ls) The idea of this script is to allow the user to check their file tree. The first if check the right number of parameters are passed and passes an error message if not the case. The first else if checks if the user directory exists and sends as error message if it does not. The second else if check if the number of arguments is equal to 2 and if so it enters another if statement. This is where the tree can be gotten so it’s where I put my semaphore to lock the user. I did not want a situation where you could write a new file to a user and request a tree at the same time. This if checks if the user directory exists and else performs an action if it does. Namely going to the user directory and echoing ok before displaying the tree. I also put in -- no report as the examples in the project seemed to have this summary supressed also. The final else statement in this script runs if it passes the previous parameter and user checks and if the user only wants to display their full user tree and not delve into any sub folders.

rm) The idea of this script is to allow the user to remove a file from their directory. The first if statements checks if the number of parameters passed is correct. The else if checks if the user directory exists and if not displays a message as such. The third else if checks if the service exists and finally if these checks all pass it enters the final else statement. I put the semaphore here as I didn’t want the user to be able to able to remove files in another window or add them while the process was happening. I use rm and the location of the file and echo a message to the user saying that the service is removed.

P) I used the p script as per the practicals whereby it locks the service based on the first parameter passed in or else if the process is already locked it enters a while loop where it prints a warning message “in while” , sleeps for one second and then tries to access the process again.

V) I used the v script as per the practicals also the if checks if it’s locked and if it is it enters the else loop where it removes the lock and exits.

Server) I used the template from the project here. I put mkfifo as my first line as I wanted it to be a case whereby as soon as the server script was called then the server pipe was created. The server pipe is deleted when the server is shutdown. Then I entered the while true loop and read all data passed to the server pipe from the client as an array called myArray. I used -r here as this maintained the integrity of any \n etc that were passed. I used a lot of indexing of the myArray to call particular parameters needed throughout the script. I used an if statement as the client specifies edit if you want to change something whereas the server has update so if edit is received from the client it changes it to update in order to work server side. I then used a case statement inside the while loop to call the relevant scripts. I also put an & at the end of each script in order for them to run in the background.

- **Init** stores the output message from running the init script and then passes this to the client pipe to be read client side.

- **Insert** takes a password from the client server and decrypts it. I used decrypt by having the clientid as the master password and decrypting the password sent from the client pipe. I then used a payload variable to convert this and the login into one string similar to how it was passed to the insert script before. I then passed all this information to the insert script and returned the output as a variable which was sent to the client pipe.

- In **show** I passed the parameter from the client pipe into show.sh and stored the resulting message as a variable. I then created a temp file and echoed the variable to this. I then grepped and split this temp file by delimiter in order to get the login and password details and removed the temp file. I encrypted the password and echoed the login and encrypted password to the client pipe.

- **Update** I passed the info from the client pipe to show.sh in order to get the current password details. I then created a temp file, cat this variable to it, grepped and split by delimiter until I had the current login and password which I passed to the (after encrypting the password) client and removed the temp file. The client then amends the details which are then read back by the server (Password encrypted). I make a temp file and cat these details to it, grep and split by delimiter and decrypt the password until I have the new login details which I store as a new payload variable. This new payload is then sent to the insert script along with other parameters and the output is stored as a variable which is sent to the client pipe and the temp file is removed.

- **Remove** I passed everything form the client pipe to the rm script and stored the output as a variable which is then echoed back to the client pipe.

- **Ls** was done exactly the same as remove previously.

- **Shutdown** When the command shutdown is sent to the server via the client it echos a message “the server has been shutdown” to the client and exits the server script.

- **Bad request** when a message that is not one of the previous valid ones is received form the client pipe it echoes a message “error bad request” to the client pipe and exits the server script.

Client) I used a case statement here but not a while loop as when it was discussed with the demonstrators they said it was not needed. The first thing I do is check for at least two parameters and if not I echo an error saying parameter issues. If there is at least two parameters it enters the else statement which makes a client pipe based on the first parameter passed to it and enters a case statement. I also create a client pipe at the beginning at remove it in every instance below.

- **Init** If parameters passed are greater than or equal to 2 then it echos the parameters to the server pipe and cats the result of the server pipe which is hopefully ‘user has been created’. Else a parameters error messages is echoed.

- **Insert** if parameters passed are greater than or equal to 3 echo to the user to type in log in and password. Encrypt the password and send to the server pipe and the subsequent message from the server is cat to the screen.

- **Show** if parameters passed are greater than or equal to 3 then pass the parameters to the server pipe and cat the results received back. Get the login in password by splitting by delimiter and decrypt the password. Echo the login and password for the user to read.

- **Ls** if parameters passed are greater than or equal to 2 the echo the parameters to the sever pipe and cat the results received back to the client pipe.

- **Edit** if parameters passed are greater than or equal to 3 then if echoes all these parameters to the server pipe. The result is then cat to a variable and echoed to a temp file. This is grepped and split by a delimiter : and the password is then decrypted. This payload is then passed to a temp file and this file is opened for the user update. The user then saves this file and the new login in and password is grepped and split by the delimiter. The Password is encrypted and sent back to the server pipe. The temp file is then removed.

- **Rm** if parameters passed are greater than or equal to 3 pass all parameters to the server pipe and cat the result back from the server pipe.

- **Shutdown** if parameters passed are greater than or equal to 2 echo parameters to the server pipe and cat the result back from the server pipe.

Challenges & Solutions

General) In general the main challenges I faced were dealing with semaphores and the placement of the P and removal V. Echoing values and reading arrays while preserving spaces and special characters also a challenge. Server.sh and client.sh were obviously the most difficult but by far the most rewarding. I struggled a lot in the practicals in general especially when trying to implement pipes, but I found working through the practicals at my own pace was very beneficial. All the tools we needed to complete the project we had covered. Below I'm listing the scripts I used and any specific challenges and solutions. I spent a lot of time before realising putting "" around variables preserved special characters ie \n. I had to do a lot of debugging which I mainly achieved through echoing results and checking if they were as expected.

Init.sh) was ok, very similar to practical's. Only issue was deciding where to put the semaphore.

Insert.sh) The challenge I found here was when trying to deal with the / that was possible in the file path. Initially I attacked this by splitting the second parameter using / as a delimiter, but this method would not work when a file had spaces without adding additional delimiter parameters by space etc so was not a good solution. The script was initially also over 100 lines with multiple if and elif so would be very difficult to implement semaphores also. The way to get this working correctly was by using dirname and basename which made the code a lot more powerful and cleaner.

I also had an issue whereby when I was passing the payload to a file it was not recognising the \n to go to a new line. The solution to this was a lot of googling but eventually found echo -e was the answer.

Show.sh) The challenge I had here was what method to print the content of the file. Initially I was going to use cat to just display all contents of the file but decided to use grep as the idea of the show was to display password and login information so I thought it would be cleaner to just look for those details specifically rather than output the contents of the entire file in case there was miscellaneous text in there.

Update) As above initially I had my insert file working by splitting delimiters / which made implementing the changes required for update very difficult. It resulted in more in a lot more conditionals making script almost unreadable. Once I found basename and dirname this update become as minor as the project suggested.

Rm.sh) similar to init.sh the main issue I had creating this script was the placement of semaphores.

ls.sh) The challenge I found here was supressing the report at the end detailing the number of directories and files. Looking through the practical's and googling was my solution.

server.sh) This was probably the one I found most challenging in the entire assignment. I was not overly familiar with arrays or case statements so found the learning curve rather steep. Similar to echo -e to recognise the \n in a string I spent a lot of time researching the fact that -r did the same for an array.

We also had the required to get all processes running in the background which I knew was achieved by using '&' however it took me a long time to figure out how to test if this was working.
As I decided to use an array the other issue I found was indexing it so the right parameters were being passed to each script. I found a lot of trial and error using this and placing miscellaneous echo statements to test this was the solution that worked best for me.
Sending information to the client pipe.
Getting to information sent to and reading from pipes was very difficult however once I spent a bit of time going over the practicals it made it a lot clearer.

Client.sh) Challenges I faced here were much the same as server. Using cat to read in from pipes was a big help as initially using read did not work. The grep -i was very useful and also using the split by delimiter.

Bonus feature

I implemented the encrypt and decrypt feature whenever the password was sent from the client.sh to the server.sh. In the project brief it stated to do this when passing the password from the client.sh to the server.sh which I did for the insert/show and update.sh scripts. However on the update the password was also being sent from the server to the client in order to be updated so I also encrypted this one. This was not as challenging as other elements of the project however my technique may not have been the most efficient. I used grep and splitting by delimiter quite extensively in this section which resulted in a lot more lines of code but I feel the result is accurate which is most important.

Conclusion

While overall I found this project difficult it was probably the most satisfying when I finally got it working and I found Anthony's advice of not getting over awed by the whole thing but rather break it into little pieces and work from there to ring true. In that regard I felt the way the project brief was structured helped greatly with this and the demonstrators were very helpful. I found the key to completing the project was studying the practical's. All the information was there and I probably could have benefited a lot more time wise by studying these more thoroughly before embarking on the project. Before the project I was wary about working in bash but it would be something I would embrace now in future.