UNIVERSITY COLLEGE DUBLIN - UCD

SCHOOL OF COMPUTER SCIENCE

Project – Password Management System

COMP30640 – OPERATING SYSTEMS

Maria Ester Braga Nunes McGarry
student number19205519

Dublin
2019

1. Introduction:

   This project contains 8 bash scripts called init.sh, insert.sh, update.sh, rm.sh, show.sh, ls.sh, server.sh and client.sh. All the scripts together create a Password Management System.

2. What is the system supposed to do:

   This system uses the script client.sh to manage users and passwords. By running that script, the user sends requests to the server.sh.

   These requests are the following:

   - init: creates new user
   - insert: inserts service into user folder. The system will ask the user for a username and password and store it.
   - show: shows the username and password for a certain client/user/ service
   - ls: shows a tree for that user, so they can see the services they have, where they are stored and any other file in that user's folder.
   - edit: allows the client to edit information in a service, such as username and password
   - rm: allows the client to delete a service
   - shutdown: allows the client to quit

   The system works by sending those inputs and outputs through pipes. The system also creates locks so when a user is being updates or removed, the system will not run any other commands for that user until the lock is deleted.

3. Architecture, design:

   The system works by sending inputs and outputs through pipes. The client will send a request to the server, and the server will send a command to execute one of the commands in each individual script. As an example, if the client wishes to create a new user, it will send a *init* request through a pipe to the server. The server will read that pipe and execute a command to run the instructions in the *init.sh* script. The instructions in this specific script is to create a new user. After the user is created, a message will be send through a pipe to be shown to the client, so the client knows if

the command was executed or not, and if not executed, what the error was. The system also creates locks so when a command is being executed for a certain user, the system will not run any other commands for that user until the lock is deleted.

4. Challenges and solutions:

- Challenge: in server: while loop keeps running trying to execute the same command over and over again. So I added read pipe_input at the end so the server will wait for input again before it keeps running.

- Challenge: how to pass arguments to each script. Found solution using:
  "${inputs[0]}"

- Challenge: in client.sh and server.sh, the server was not reading the inputs correctly from pipe because of the spaces between the inputs, example:
  echo $client_id $command $user >$server_pipe
  So I used | to separate the inputs to send to pipe:
  echo "$client_id|$command|$user" >$server_pipe
  server.sh then uses IFS to read the spaces:
  IFS='|' read -ra inputs <<< "$pipe_input"

- Challenge: init, insert, update, rm scripts should only echo “user created”, “service created”, “service updated” “service removed” if the program does each one of those things successfully. So I used $? To show result of previous command to check if that command was executed:

```
if [ $? -ne 0 ]; then
        exit 1
fi
```

  Only after the check, will the script echo the statements above for each script, meaning they were successful.

- Challenge: in update.sh script I tried creating a folder and a file simultaneously, if the folder doesn’t exist, then I have the following error:

(base) Esters-MacBook-Air:server esternmcgarry$ ./insert.sh user1 Bank/aib.ie "password: mypassword\nlogin: myLogin" ./insert.sh: line 22: user1/Bank/aib.ie: No such file or directory

The solution I found was to use mkdir -p, this allows to create multiple directories in one command.

- Challenge: when executing ls from client.sh, only the first line of the tree would echo. So I used base64 command to code and decode.

- Challenge: scripts that create a file that is going to be used only temporarily, like a pipe or a lock, need to delete that pipe when it exits. But if the script doesn't run to completion, the pipe will not be delete. Solver the problem using a trap at the beginning, this way the file will get deleted, no matter what kind of exit.

  trap **"rm -f** $user**/**$user**.lock"** EXIT

  trap **"rm -f** $client_pipe**"** EXIT

  trap **"rm -f** $server_pipe**"** EXIT

- Challenge: needed to execute a command and put that command into a variable. Solved by using variable=`command`: payload=`cat $tempfile`

- Challenge: I ran all the scripts in the terminal in my Mac, then transferred them to test in UCD server using, for example:

  scp ./server.sh cs19205519@csserver.ucd.ie:~/COMP30640/bash_project

- Ran all the tests in the UCD server, here are the main ones:

  cs19205519@csserver:**~/COMP30640/bash_project**$ ./client.sh client1 init user1
  OK: user created
  cs19205519@csserver:**~/COMP30640/bash_project**$ ./client.sh client1 insert user1 bank
  Please write login: htr
  Please write password: dfg
  OK: service created
  cs19205519@csserver:**~/COMP30640/bash_project**$ ./client.sh client1 edit user1 bank
  OK: service updated

```
cs19205519@csserver:~/COMP30640/bash_project$ ./client.sh client1 show user1
bank
login:htrnpassword:dfg!!!
cs19205519@csserver:~/COMP30640/bash_project$ ./client.sh client1 rm user1
bank
OK: service removed
cs19205519@csserver:~/COMP30640/bash_project$ ./client.sh client1 show user1
bank
Error: service does not exist
cs19205519@csserver:~/COMP30640/bash_project$ ./client.sh client1 ls user34
OK: user34 ├── bank └── user34.lock
```

5. Conclusion:

   This Password Management System project presented many challenges as I was writing the scripts, there was a lot of research involved to find solutions for the problems and there was a lot of back and forth changing scripts. One of the most important lessons for me was learning that using echo in different part of the scripts to see which part is not working and this way narrow down the problem areas is essential for debugging.

6. Bonus (extra feature):

   If a file lock exists, a command sent from client to server will not be executed and the programme will exit. So the program tries a few more times before it exits, I created a system based on the code found here:

   https://www.kiloroot.com/bash-two-methods-for-job-control-simple-lock-files-and-flock/

   and here:

   https://www.tutorialspoint.com/unix/until-loop.htm

   This way the lock system will check again if the lock file exists every second for 10 seconds. If the lock is removed before the 10 second limit, then the command will be executed normally.

   I adapted the code to fit what I wanted in my scripts, so my lock works as follows:

```
# check if lock exists
if [ -f "$user/$user.lock" ]; then
```

```
   seconds=0   #if locked wait 10 seconds, checking every second if lock removed
   while [ -f "$user/$user.lock" ]
   do
     sleep 1
     # increment seconds waited by 1
     seconds=`expr $seconds + 1`

      if [ $seconds -eq 10 ]
      then
        echo "timed out waiting for file"
        exit;
      fi
   done
else
   touch "$user/$user.lock"
   trap "rm -f $user/$user.lock" EXIT   #if script creates lock, always remove it, no
matter what exit type
fi
```