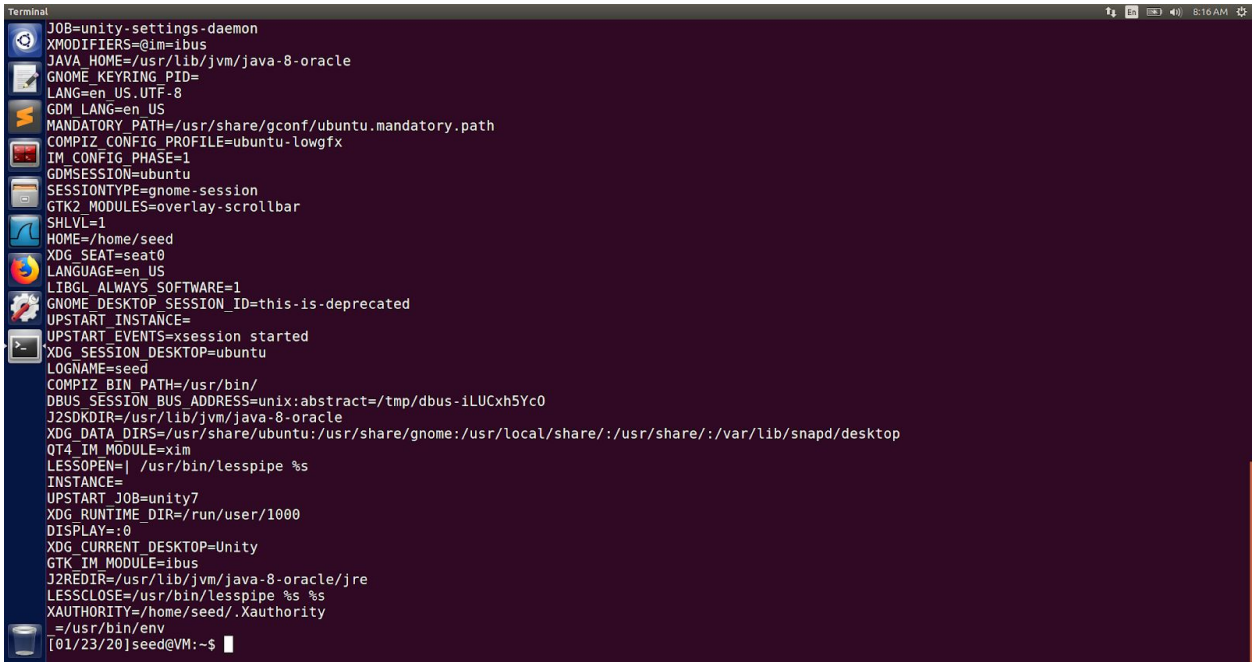
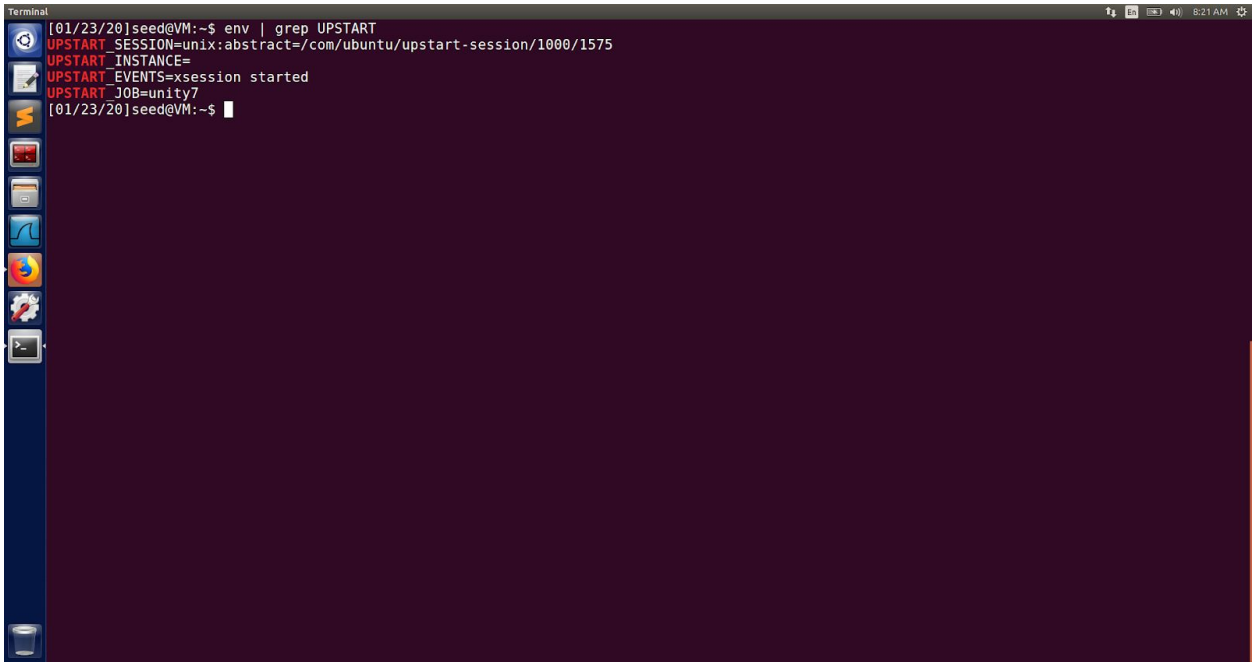
Task 1 : Manipulating Environment Variables

Output when the command ***env*** is used to print all the environment variables:

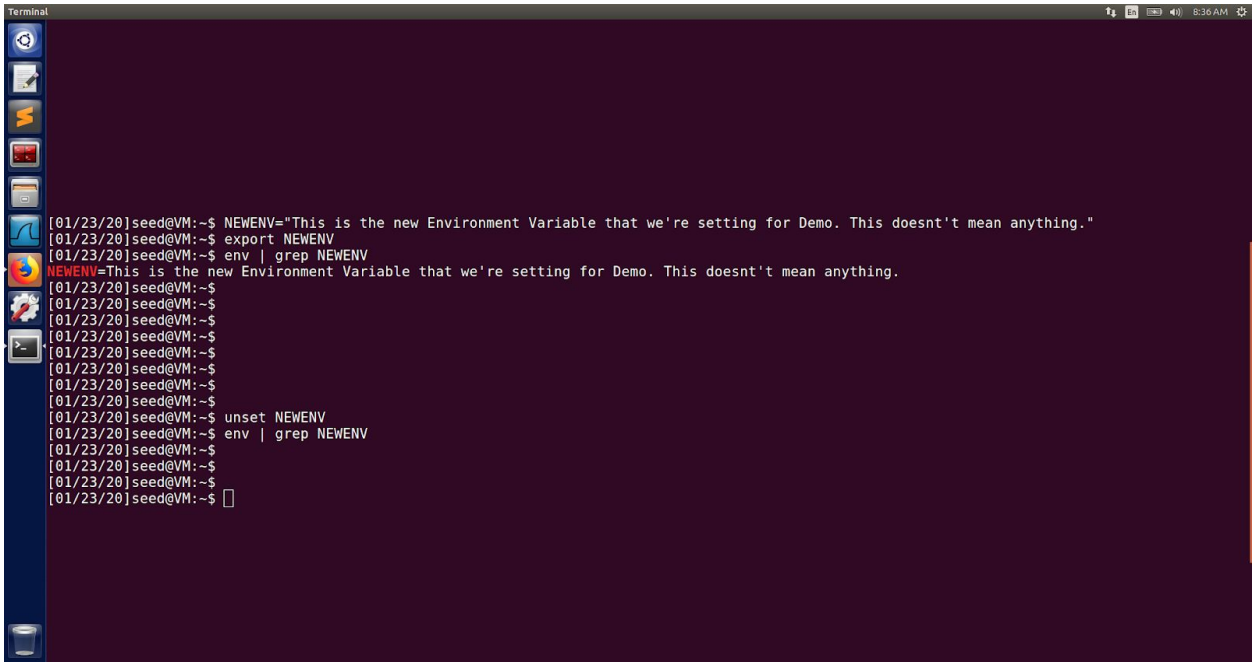


Output when we execute grep command to search for a particular command containing the relevant search keyword:



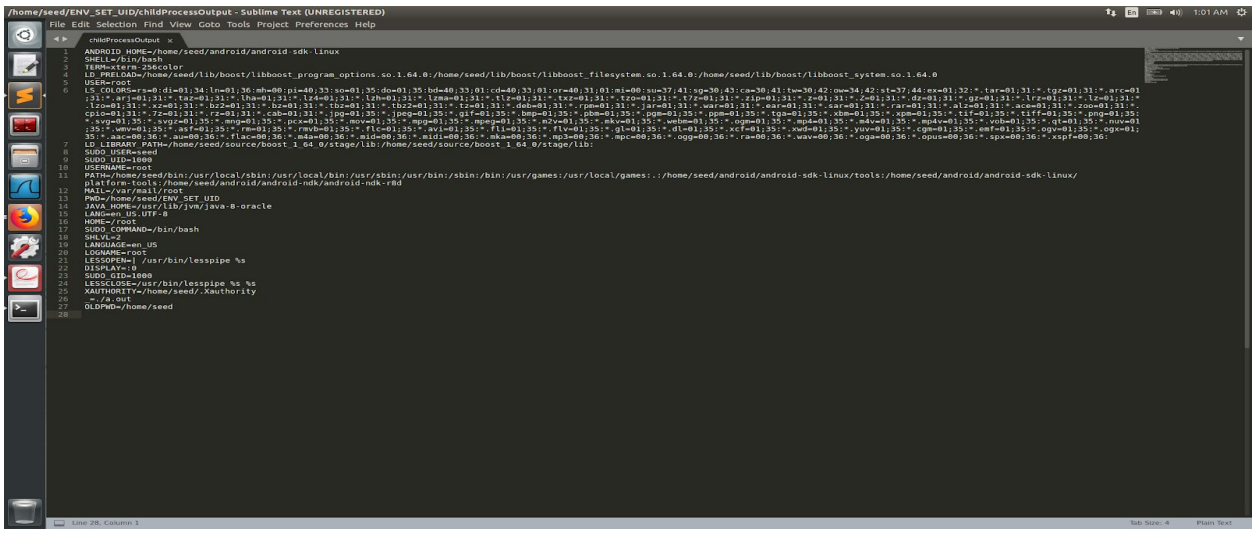
Setting and Unsetting an Environment Variable.

Here, we're setting a new environment variable using the export command and unsetting the same using unset command. It is cross verified using the grep command on env with the variable name we've set.

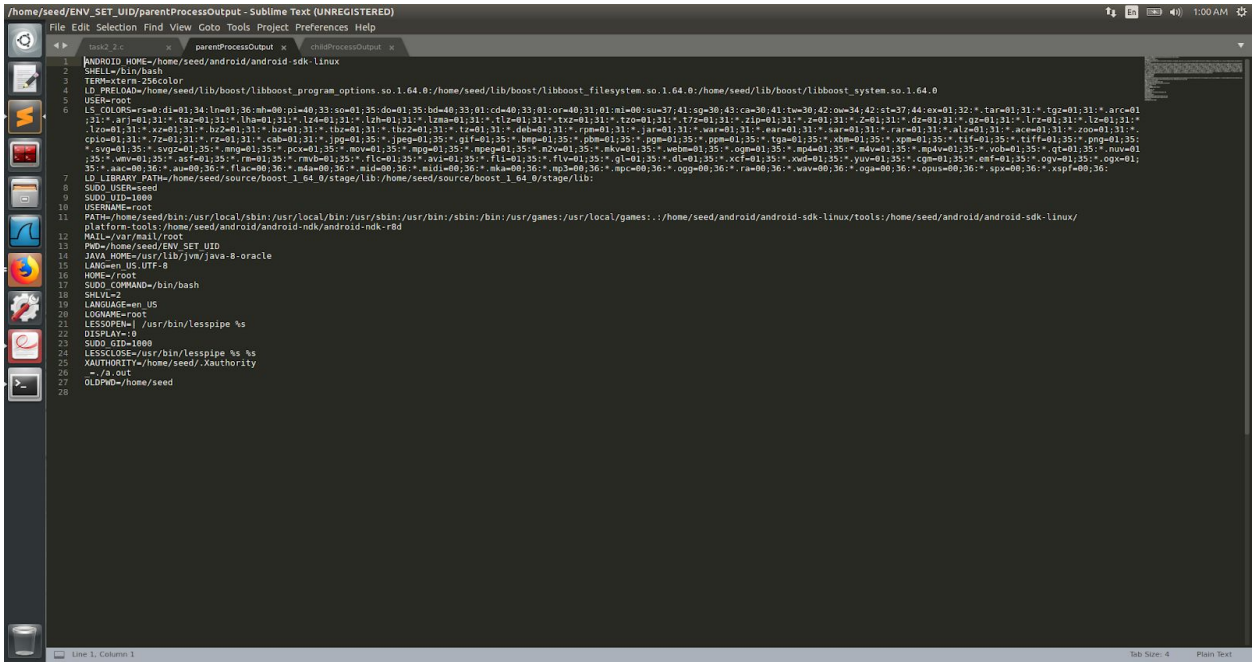


Task 2*: Passing Environment Variables from Parent Process to Child Process*

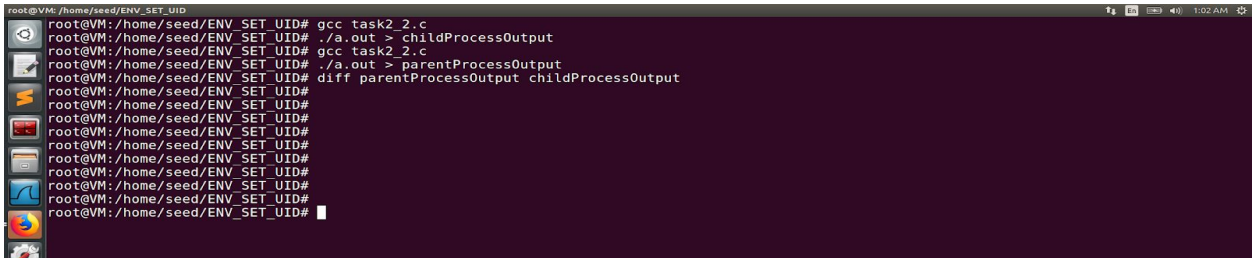
- **Step 1:** We execute the given code and the output is the list of all the environment variables.



- **Step 2:** Here we execute the parent process and from the output it is evident that the child process inherits the environment variables from the parent process.



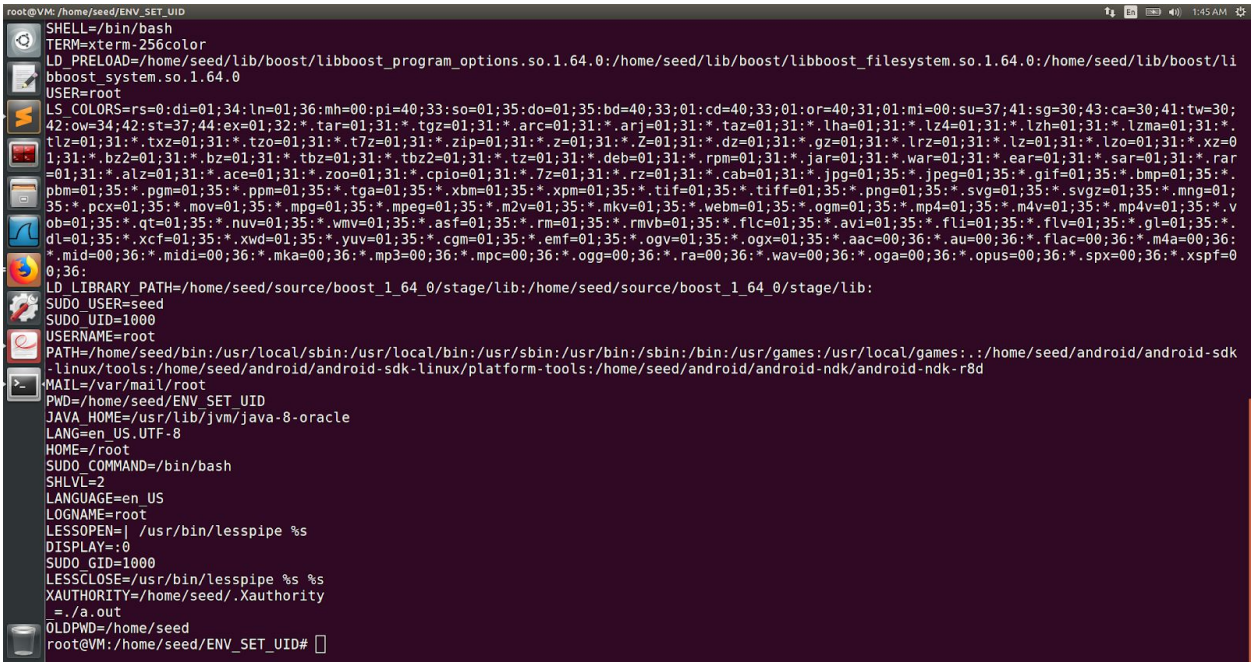
The diff command on both the files returns null. This proves that both the files are the same.



Task 3*: Environment Variables and execve() :*

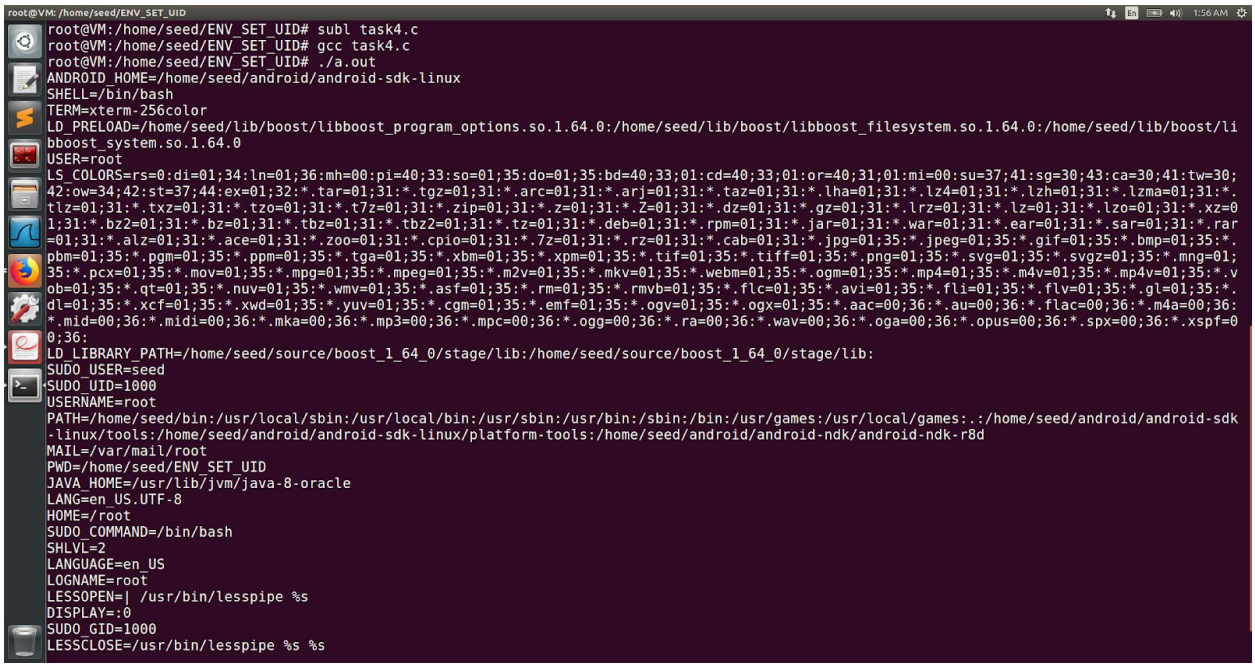
- **Step 1:** Since the third argument passed to the execve() function is null, we cannot see any output.

- **Step 2:** With the environ string passed to the execve() function as ***execve("/usr/bin/env",argv,environ)*** the name of the executable file after compilation is added to the environment variables list at the end. It's **a.out** in our case.



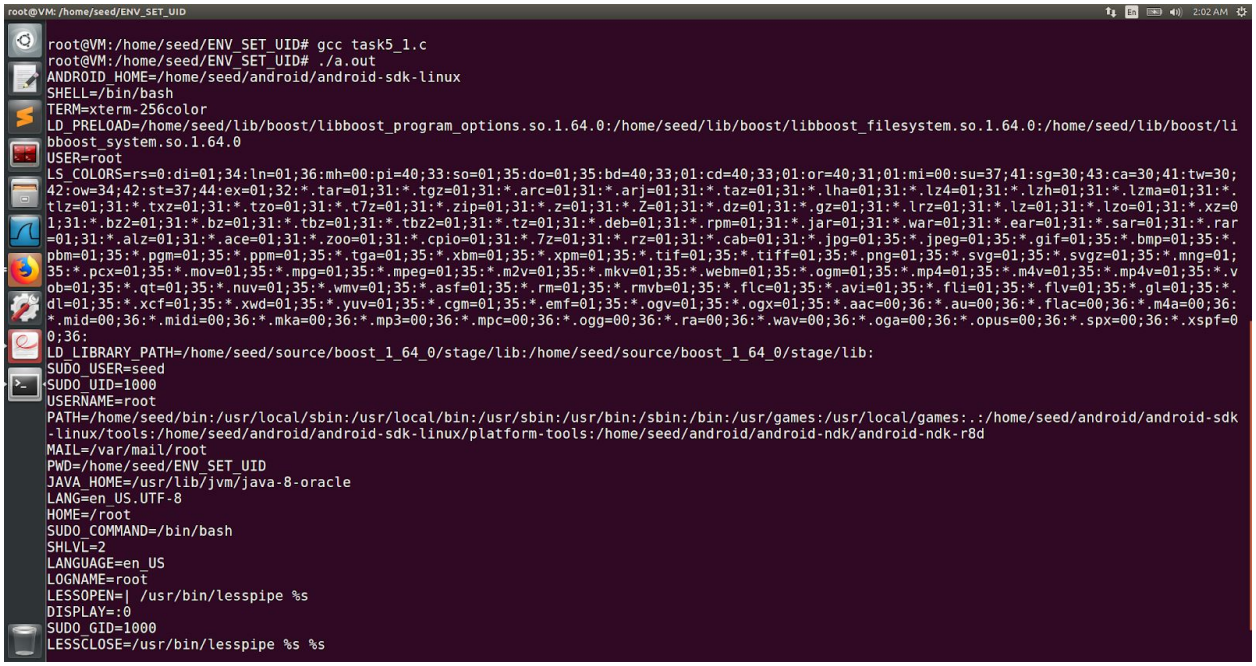
Task 4: Environment Variables and system()

On execution of the system(), it first spawns the bash and bash executes a command that prints all the environment variables.

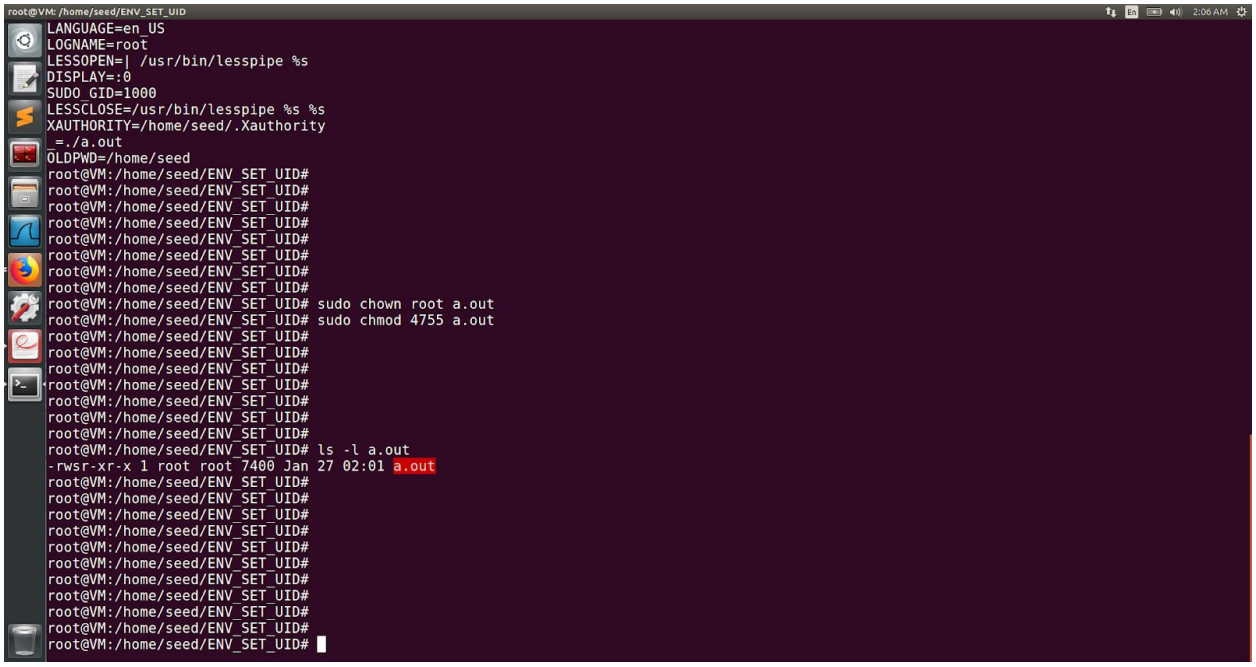


Task 5: Environment Variable and Set-UID Programs

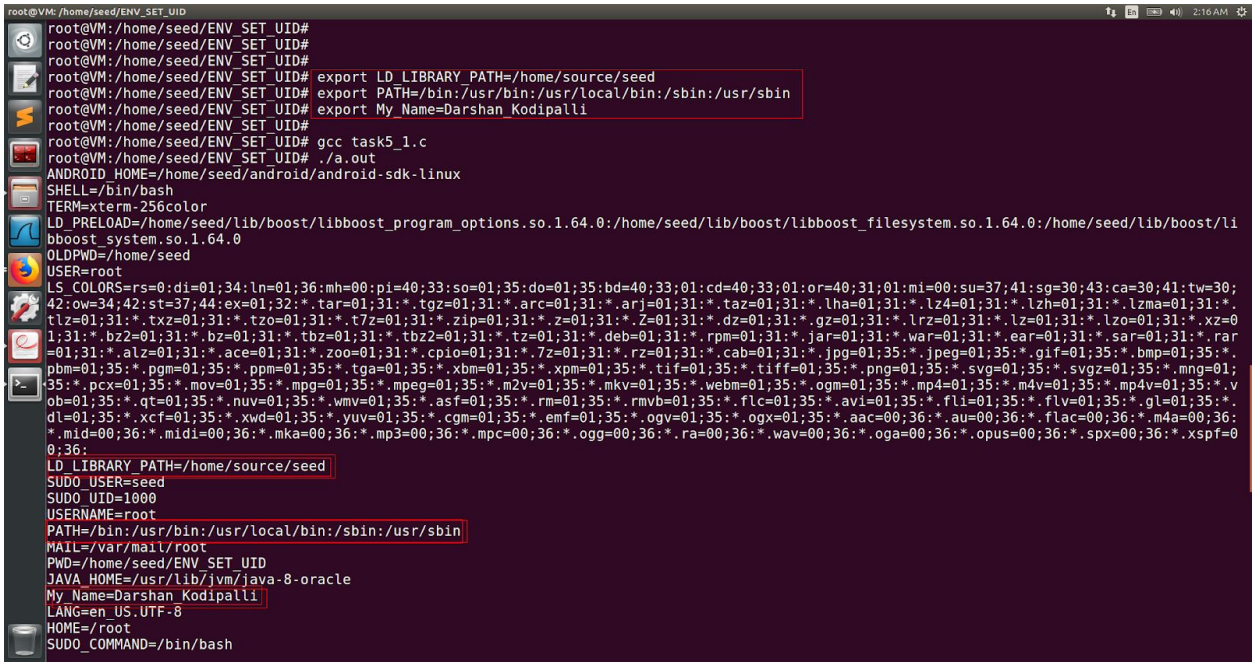
- **Step 1:** On execution of the output file after compilation, the program prints all the environment variables.



- **Step 2:** The output after changing the ownership of the executable and changing the permissions. We can see the owner is now the **root,** and it's permission also being changed.



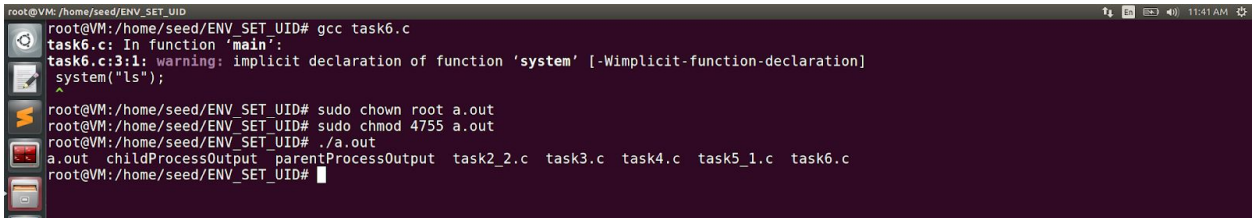
- **Step 3:** After setting the environment variables using the export command and re-running the code we can see all the environment variables we set. All the environment variables set in the parent process(shell) got into the set-uid chile process. The output shows the environment variables we set previously.



Task 6: The PATH Environment Variable and Set-UID programs.

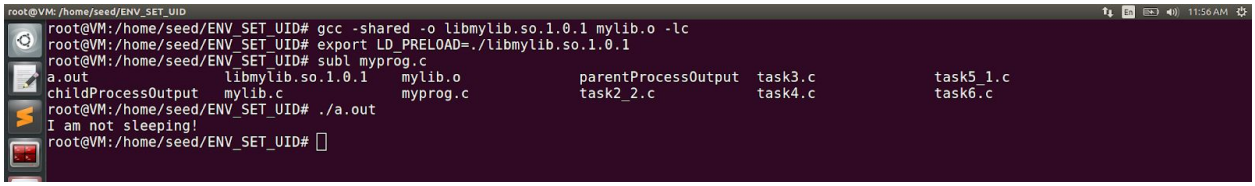
Can you let this `Set-UID` program run your code instead of `/bin/ls`? If you can, is your code running with the root privilege? Describe and explain your observations.

→ We can let this Set-UID program run seamlessly but as the system can be exploited, it is risky. Since there are some privileges changed to the program, the code is now running with the root privilege. Since the system() spawns the shell which inturn executes the command that we run, it can easily be exploited and misused.

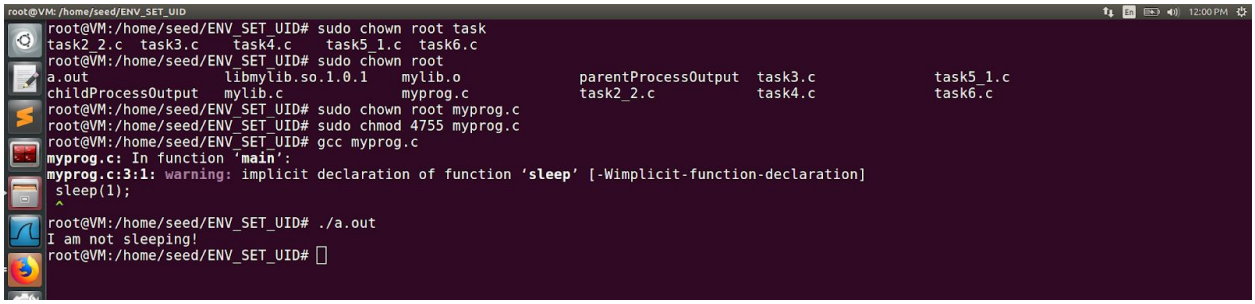


Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

- **Step 1:** Run it as a regular program as a normal user.



- **Step 2:** Make it a Set-UID Program and run it as a normal user.



- **Step 3:** Changing the program as a Set-UID program, running it as a root, Exporting the LD_PRELOAD environment variable to the root and rerunning it.

- **Step 4:** With another user account userB, changing myprog to a Set-UID userB program, export the LD_PRELOAD variable to userB's account and re-running it.

Task 8: Invoking External Programs using system() versus execve()

- **Step 1:** If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?
  → Since we're making use of a system() which calls the shell to execute the command and shell has root privileges so we will be able to remove a file. Therefore we can compromise the integrity of the system.

- **Step 2:** Do your attacks in Step 1 still work? Please describe and explain your observations.
  → No, the attack will not be successful, since we're making use of execv() which does not create a new proess, instead it replaces bash with command to be executed. Since the **cat** command is used, we get a similar output when run both the versions. More exploitation can be done with the system() than with the execve().

Task 9: Capability Leaking

Run the program as a normal user, and describe what you have observed. Will the file `/etc/zzz` be modified?

→ Since the file **/etc/zzz** is open even before the Set-UID, we can see the file has been modified. To avoid this, we can move Set-UID(setuid(getuid())) above the open().