Babaiah Gandluru

R11899782

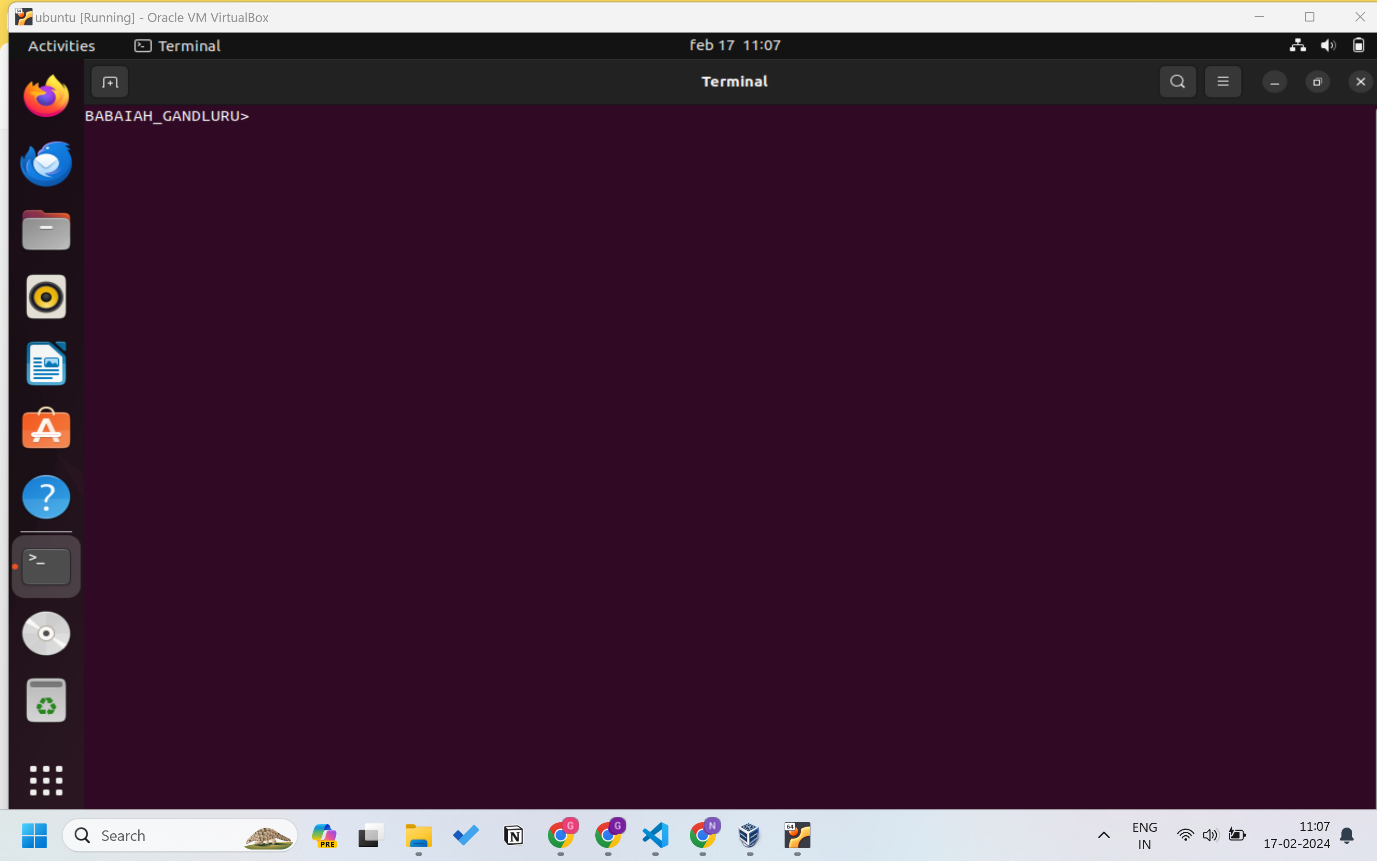
Introduction to Information and Computer Security

(CS-5340-001)

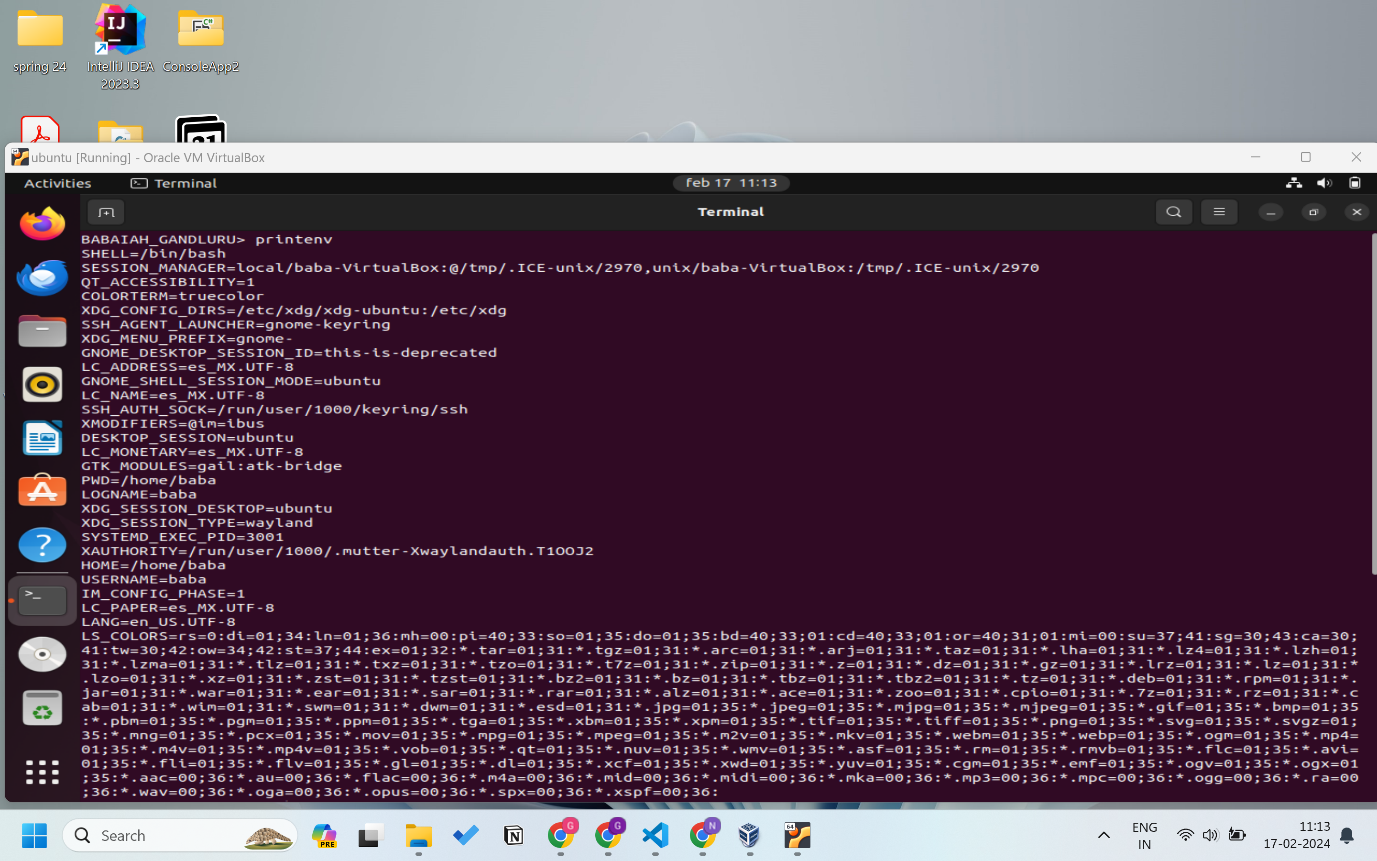
Environment Variable and Set-UID Program Lab

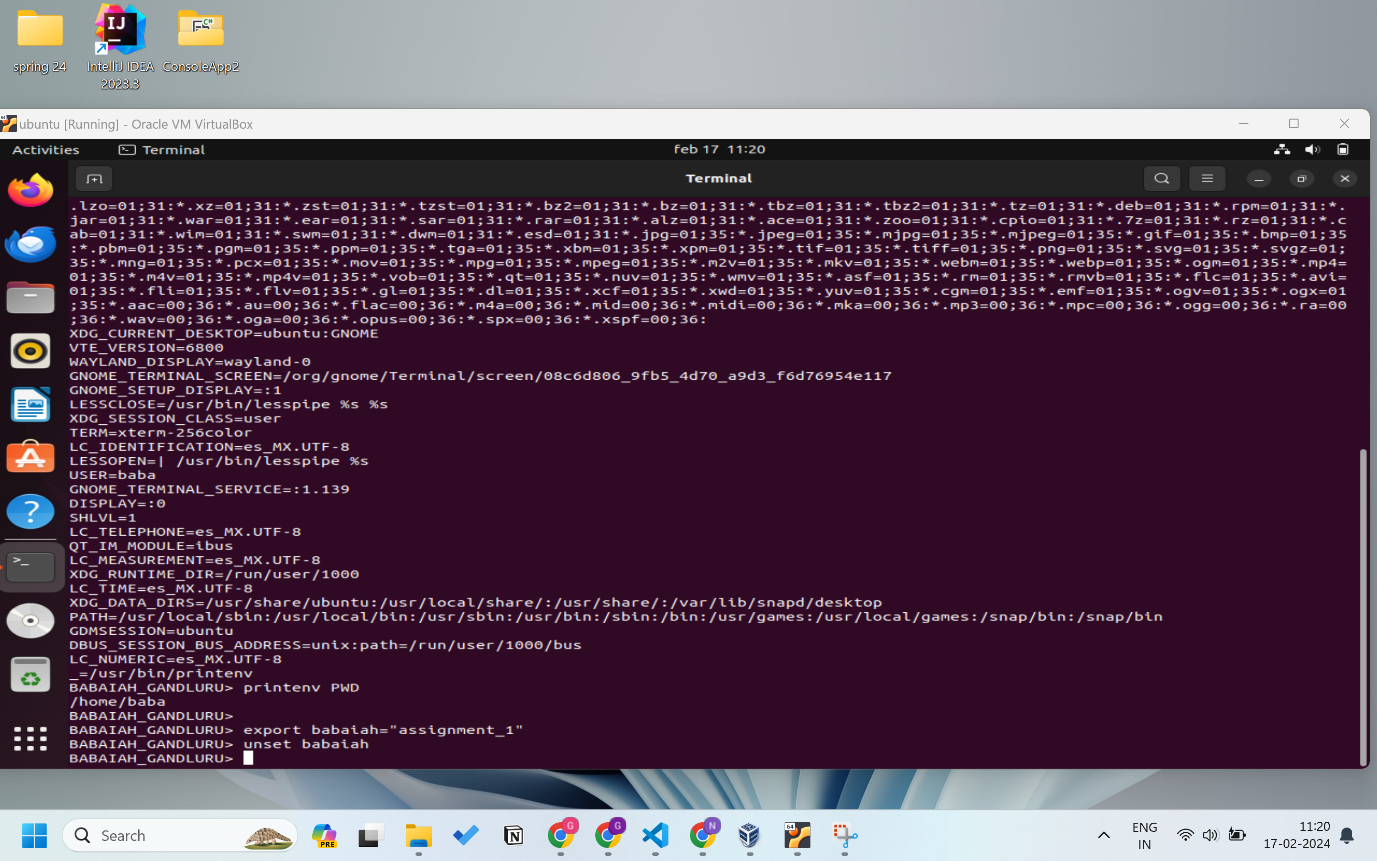
Assignment-1

I executed all 9- tasks and explained them



***Task-1***

First I executed the command printenv to display the current environment variables in the terminal



Then I executed printenv PWD this command prints the value of the PWD variable in my virtual machine it is /home/baba

Then I executed commands like

export babaiah=”assignment\_1”

unset babaiah

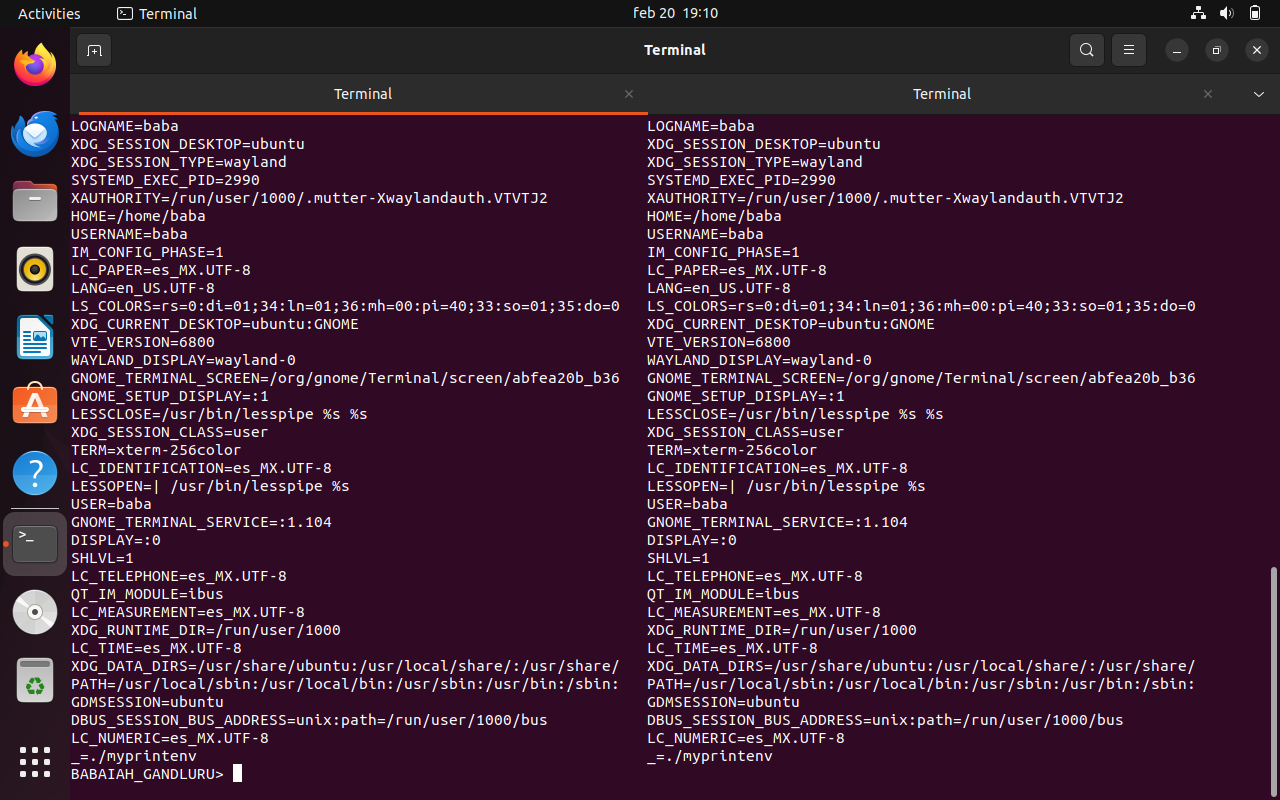
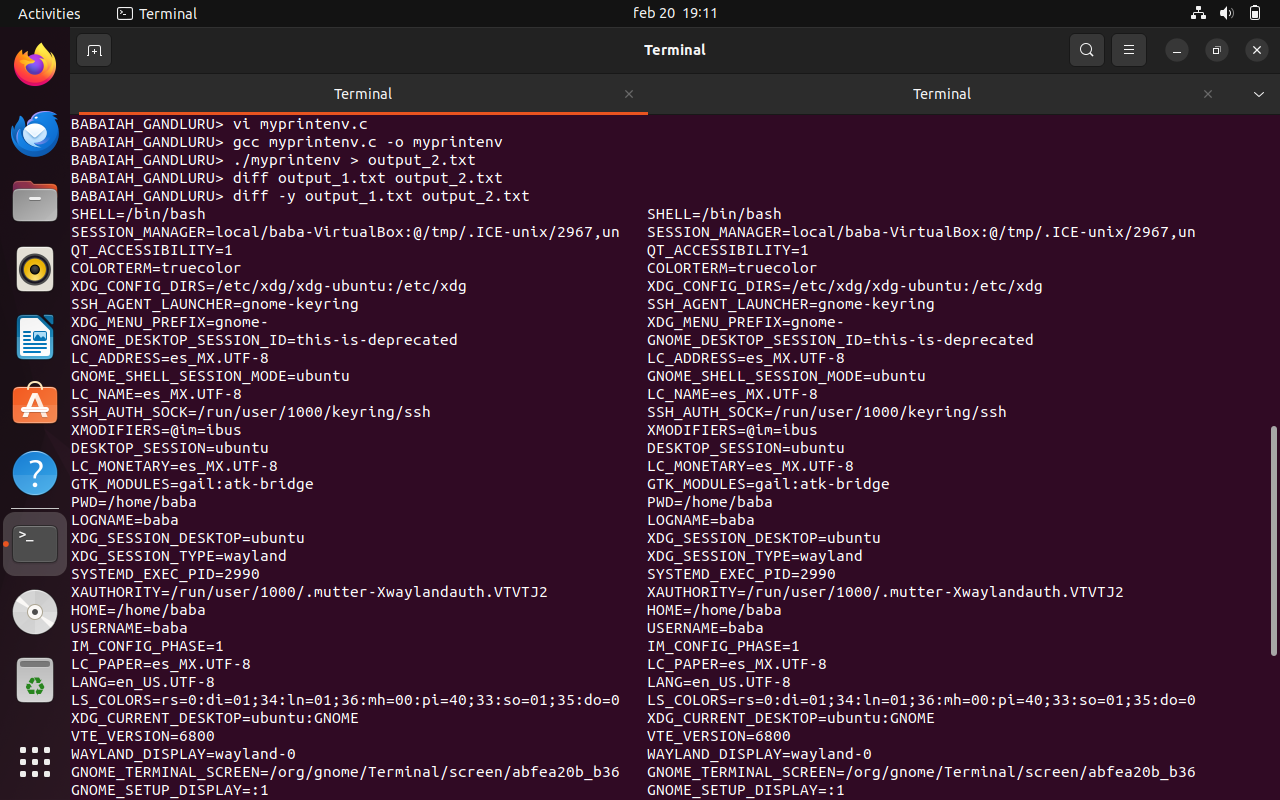
export is used to define a new environment variable (or) to update the value of an existing one.

unset is used to remove an environment variable from the current shell session.

so i defined a new variable babaiah and assigned it value "assignment\_1"

and removed it with unset command

***Task-2***



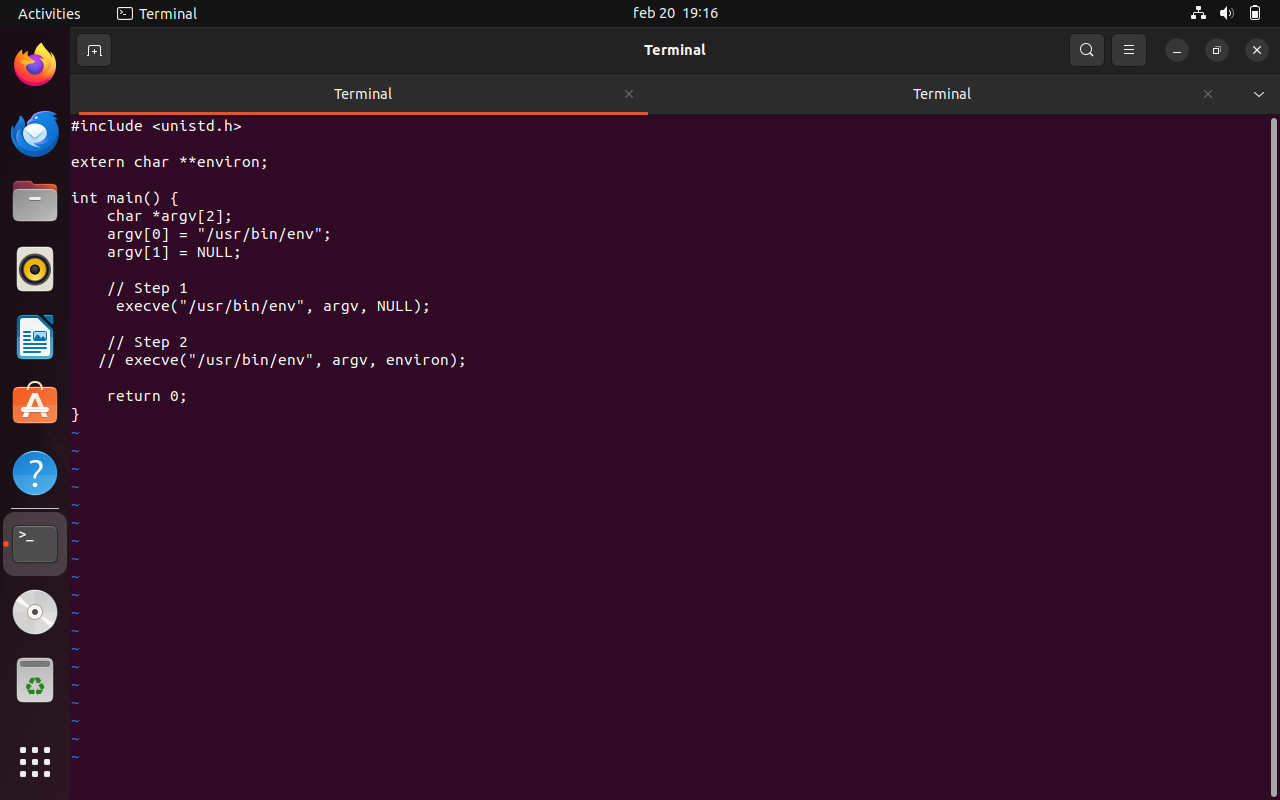
I executed both parent process and child process in output\_1.txt file and output\_2.txt file and then I executed diff command to see the differences in the outputs

I dont see any differences in the outputs both outputs are identical

So, In conclution

We can say that in systems like Unix, environment variables are inherited by child processes from their parent processes. This inheritance will allow child processes to access and utilize the same set of environment variables as their parent processes without explicitly passing them.

***Task-3***

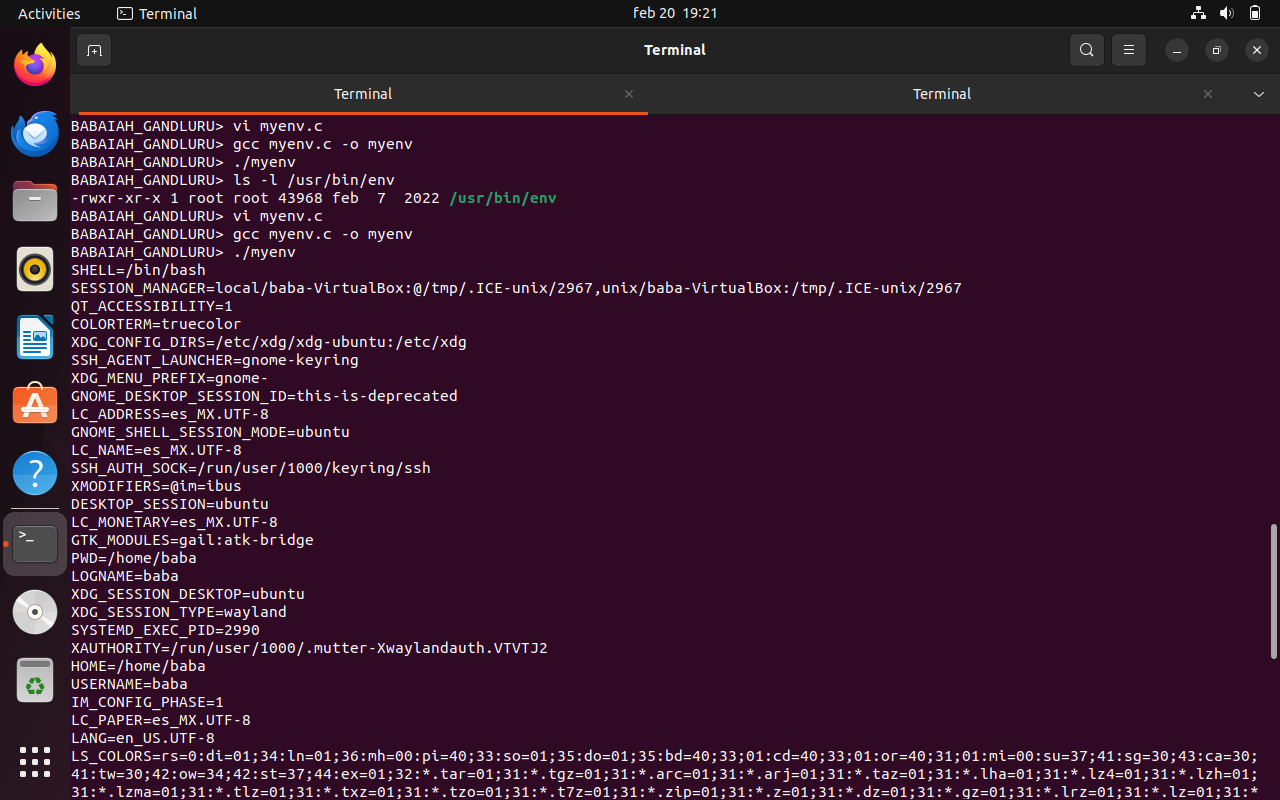


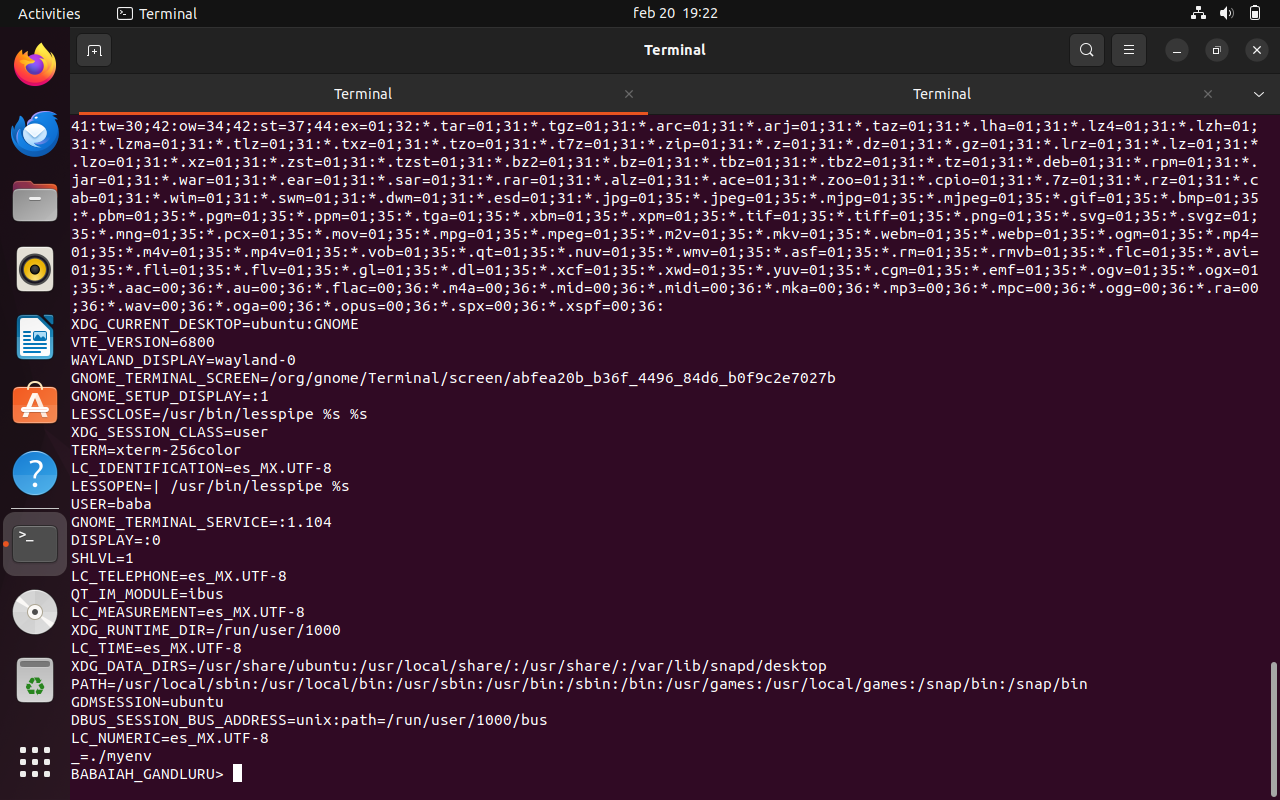
I executed both codes with “NULL” and “environ” arguments

When NULL is passed as the envp argument the new program /usr/bin/env does not inherit the environment variables from the current process.

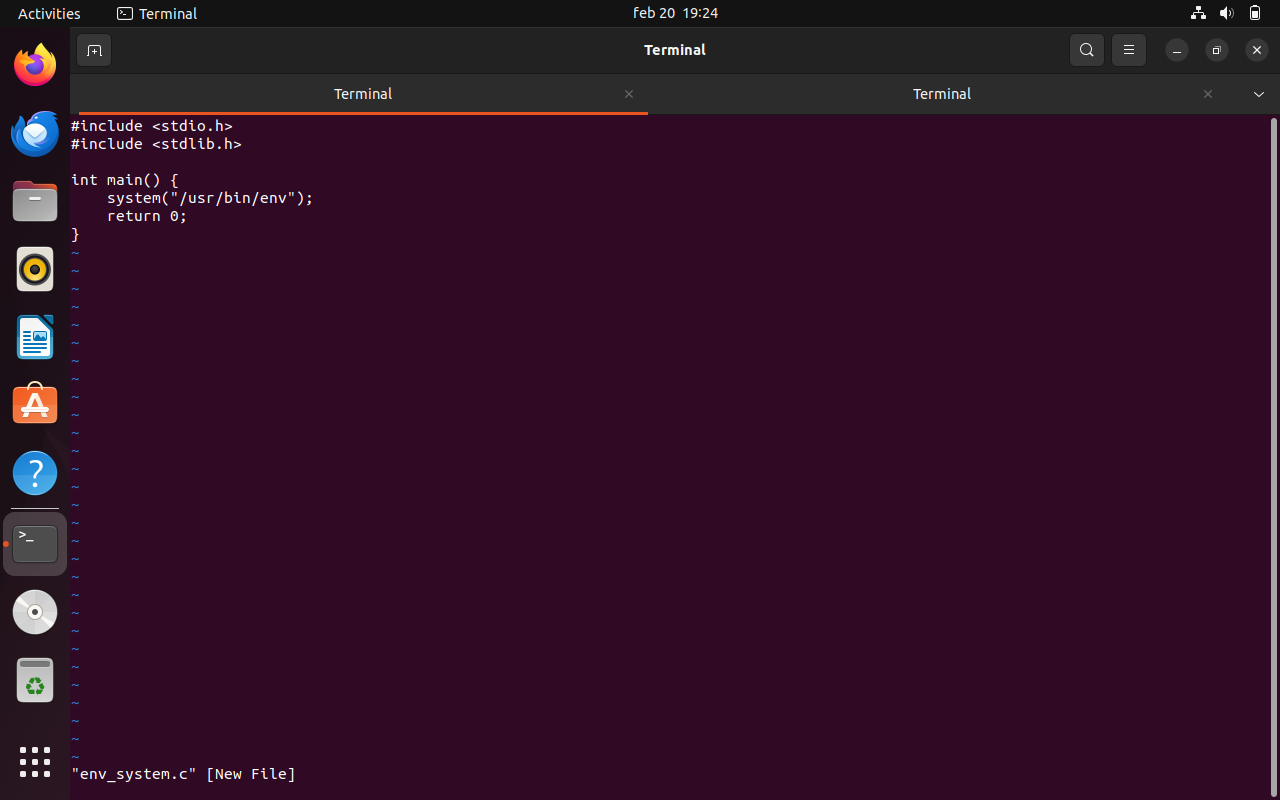
But when we pass environ as the argument to execve(), the current environment variables are passed to the new program /usr/bin/env. I got list of environment variables as shown in the screenshort

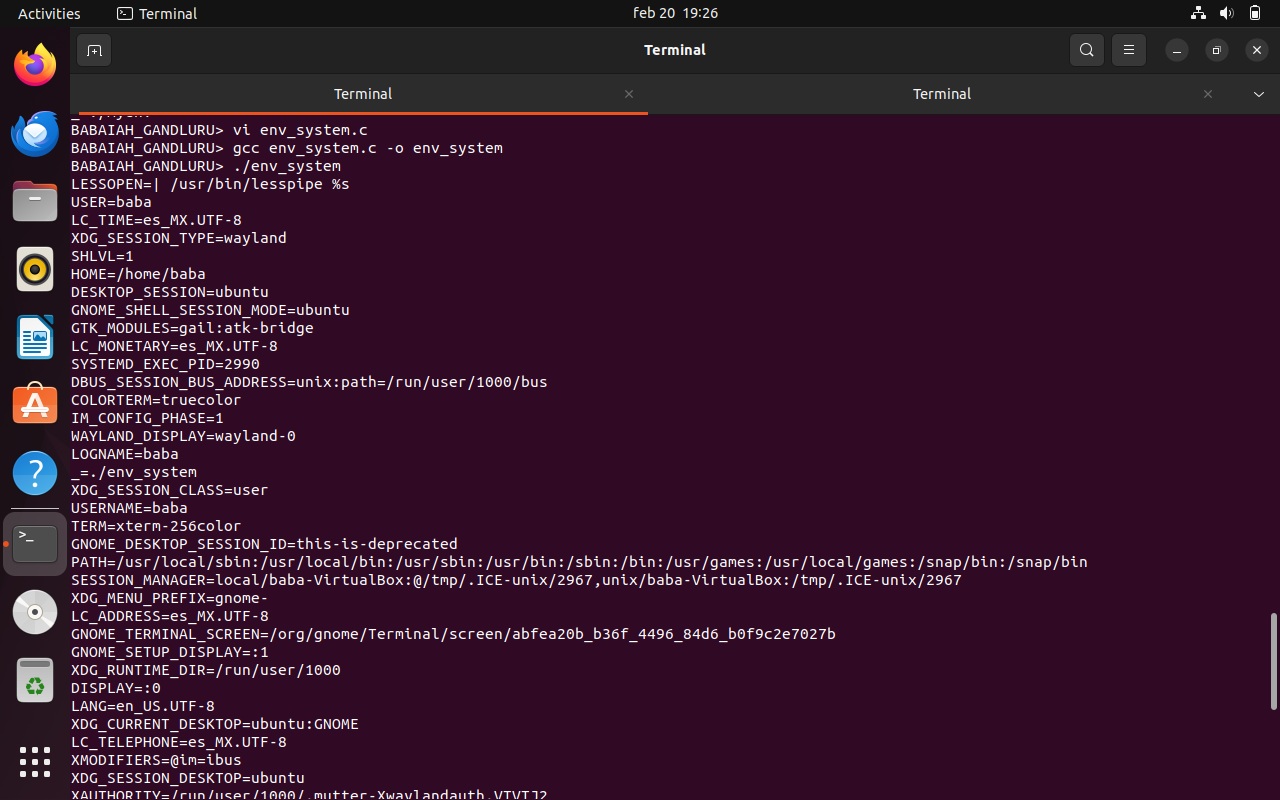
so, we can conclude that the new program inherited the environment variables from the current process.





***Task-4***



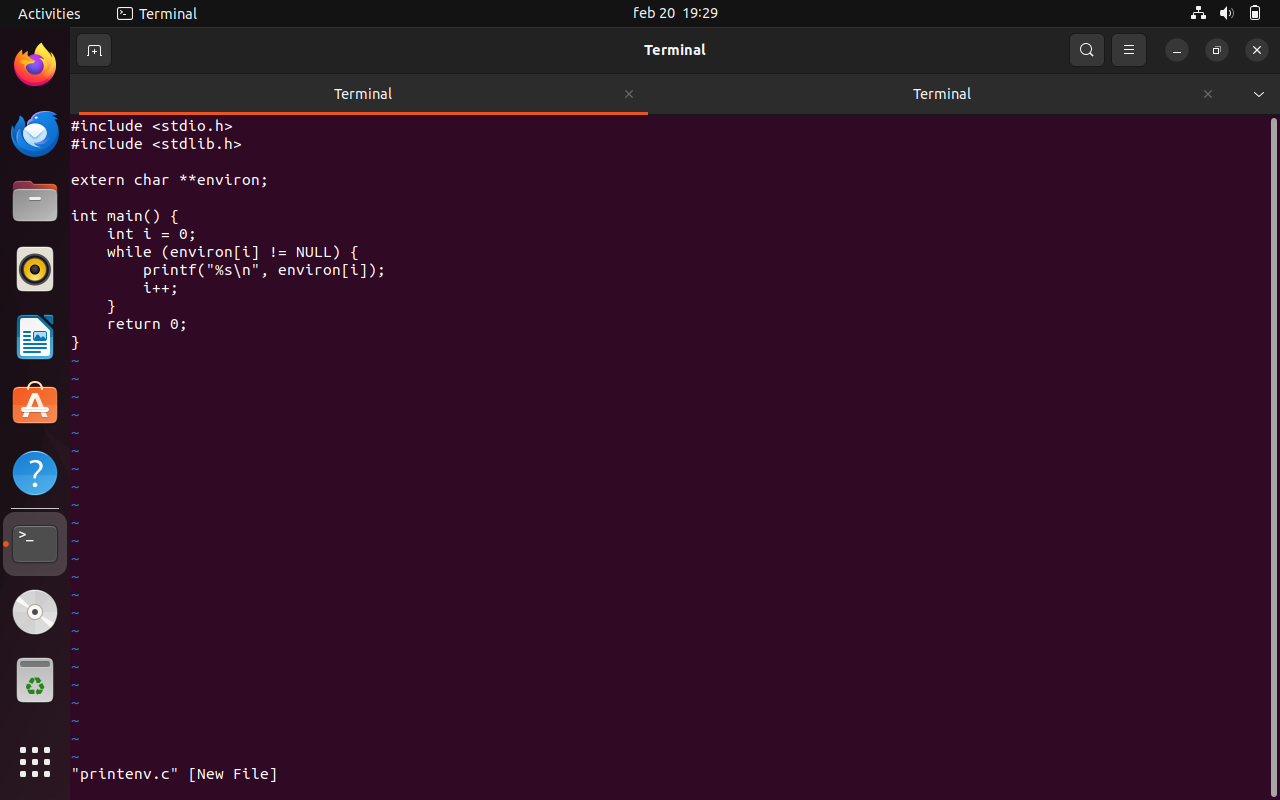


I executed this code and the output of env command executed by /bin/sh includes the environment variables of the calling process

so, we can conclude that the system() function in the code executes command by invoking /bin/sh

This is similar to using execve() to execute the command and inherit the environment variables

***Task-5***



I executed the code and following commands

sudo chown root foo

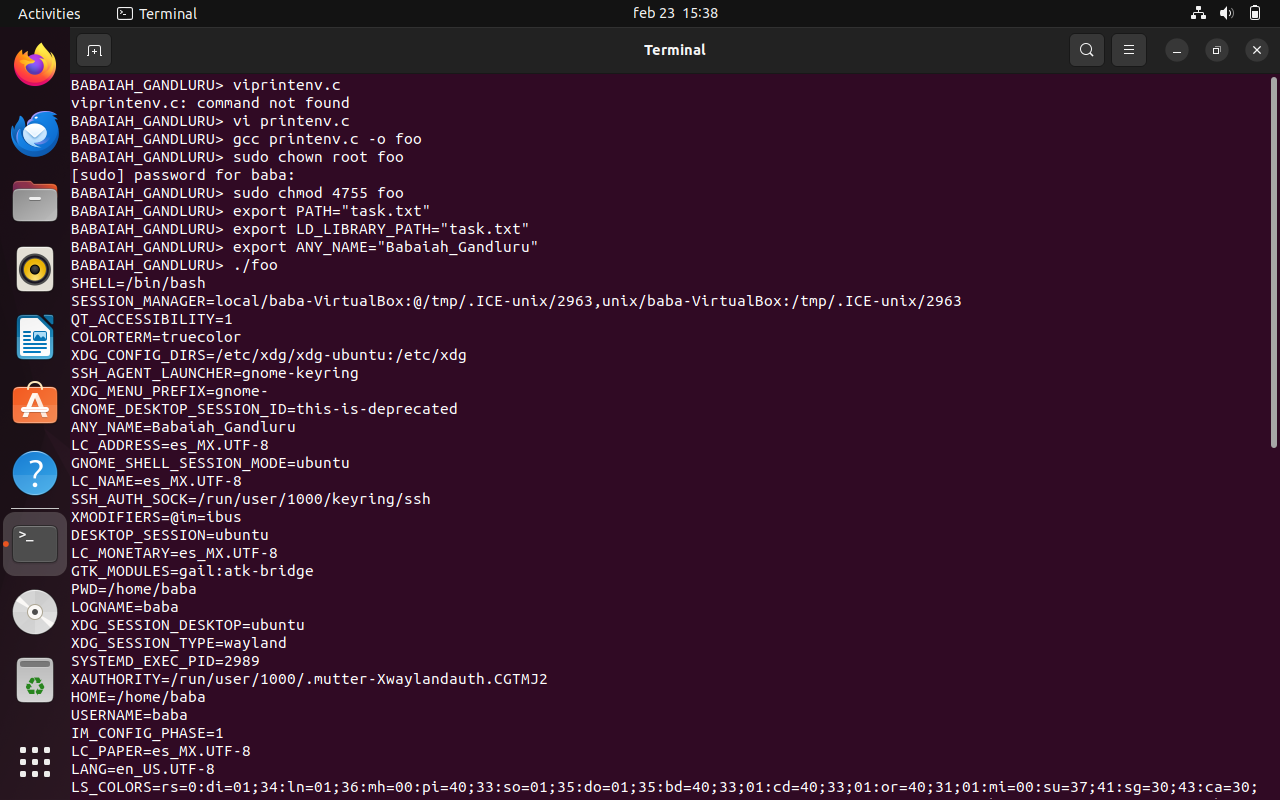
sudo chmod 4755 foo

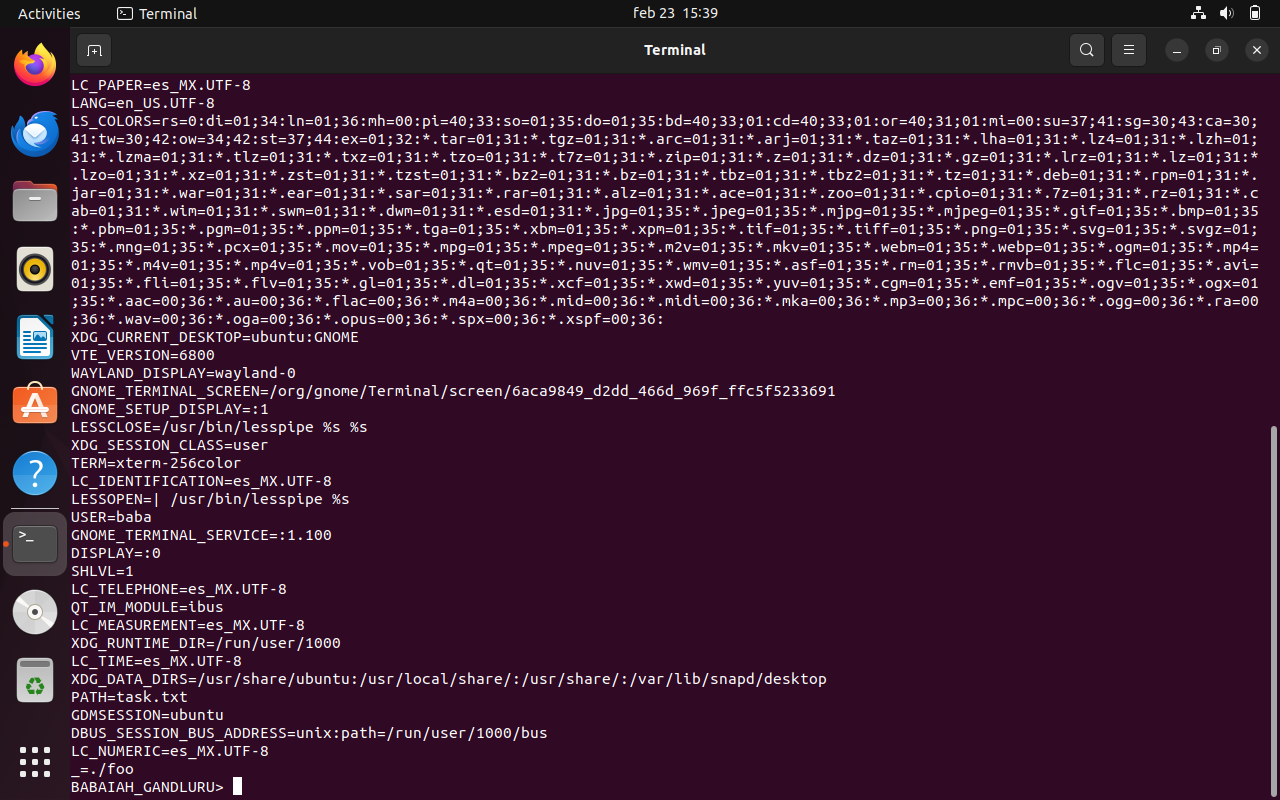
export PATH="task.txt"

export LD\_LIBRARY\_PATH="task.txt"

export ANY\_NAME="Babaiah\_Gandluru"

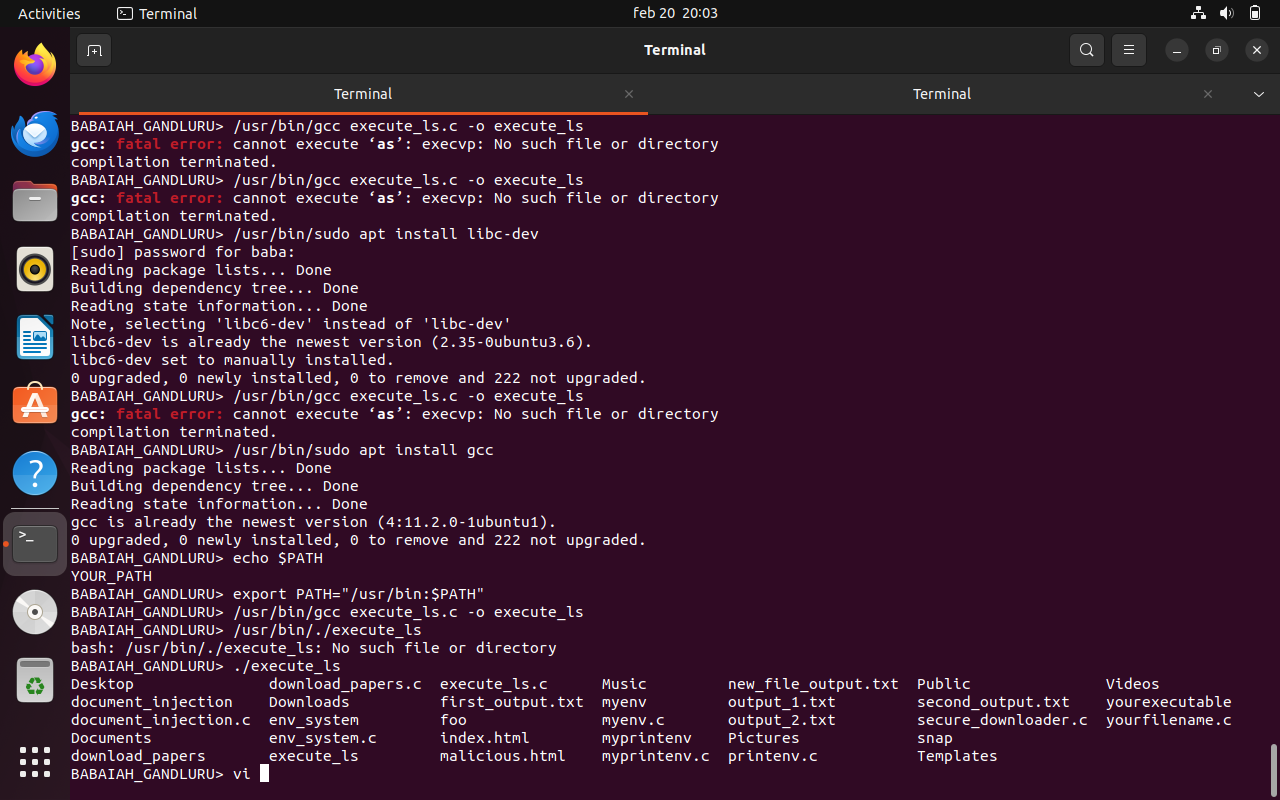
./foo

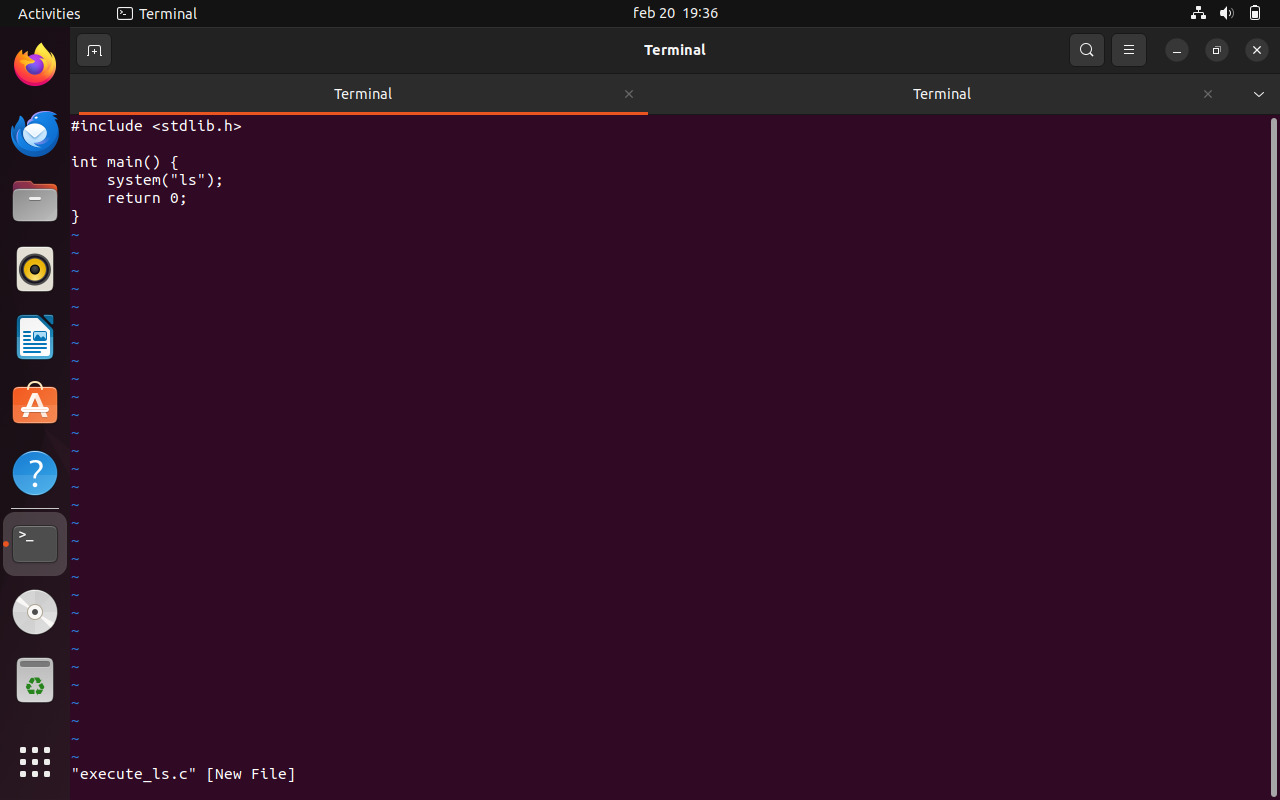




After running the Set-UID program foo, I observed that it gave me all the environment variables that were set in your shell process, including PATH, LD\_LIBRARY\_PATH, and ANY\_NAME which is Babaiah\_Gandluru. This conforms that environment variables set in the user's shell process are inherited by the Set-UID child process.

***Task-6***



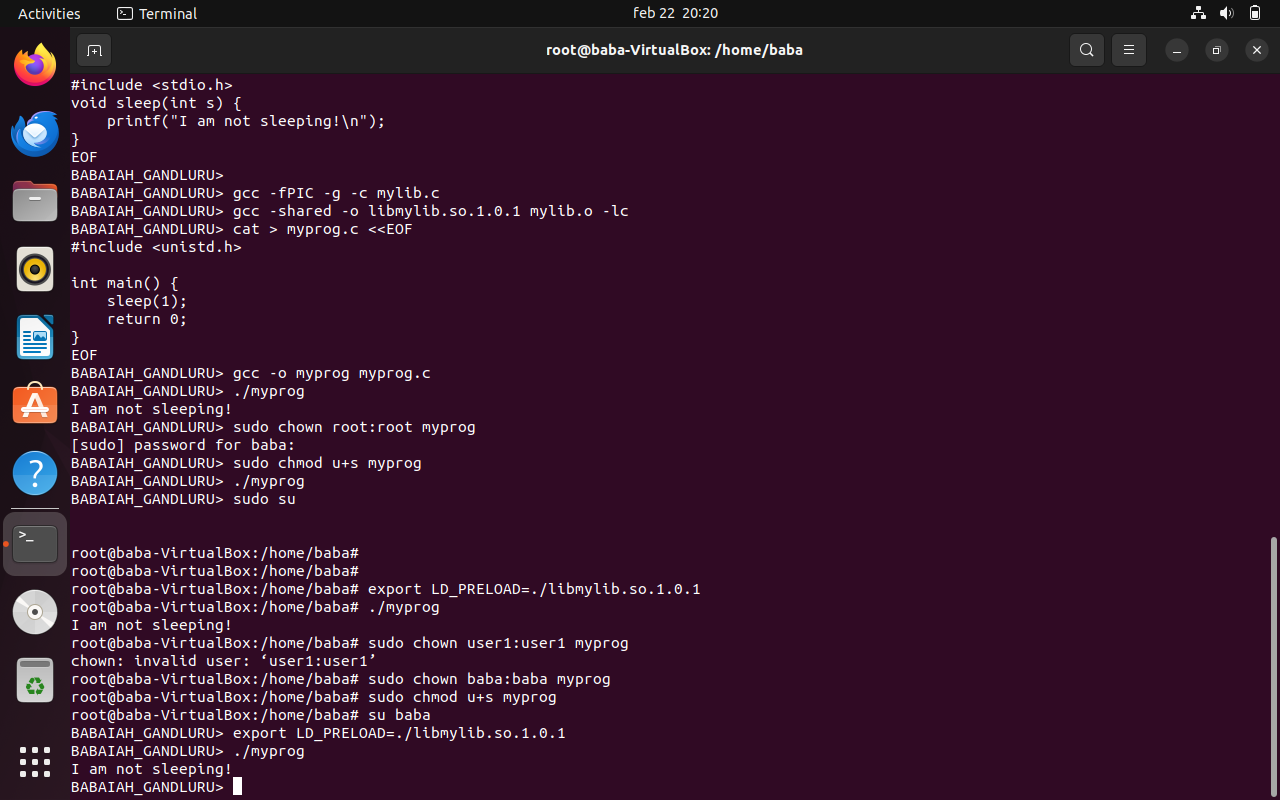


After running the Set-UID program execute\_ls, I observed that it executes the "ls" command and lists all the contents of the current directory.

Since the program uses a relative path for the ls command (system("ls")), it relies on the shell's PATH environment variable to locate the ls executable.

I learnt that this can allow Malicious users to exploit by modifying the PATH environment variable to point to a different directory containing a malicious executable named ls. and When the Set-UID program is executed by the host user it would run the malicious code instead of /bin/ls.

***Task-7***



I executed both myprog.c and mylib.c codes

I executed the myprog.c in 4 scenarios

1. when I executed myprog as regular program and as a normal user.

I got output as

I am not sleeping !

1. when I executed myprog as Set-UID root program and as a normal user.

I didnt get any output for this scenario

1. when I executed myprog as Set-UID root program and as the root account.

I got output as

I am not sleeping !

1. when I executed myprog as Set-UID "baba" program (i.e., the owner is user1, which is another user account).

I got output as

I am not sleeping !

The difference in the behavior is due to the LD\_PRELOAD environment variable interactions with Set-UID programs. When LD\_PRELOAD is set it loads the specified library before all other librarys.

But,in the case of Set-UID programs there are some restrictions imposed by the operating system for security reasons.

When I executed myprog as a regular program the LD\_PRELOAD command worked as expected and the sleep() function from mylib.so is overridden and called.

I got "i am not sleeping !" message

But, when I executed myprog as a Set-UID root program:

In case if LD\_PRELOAD is not set again within the program the LD\_PRELOAD environment variable is not considered due to security restrictions. so the original sleep() function is executed in this case and

I didnt get "i am not sleeping !" message

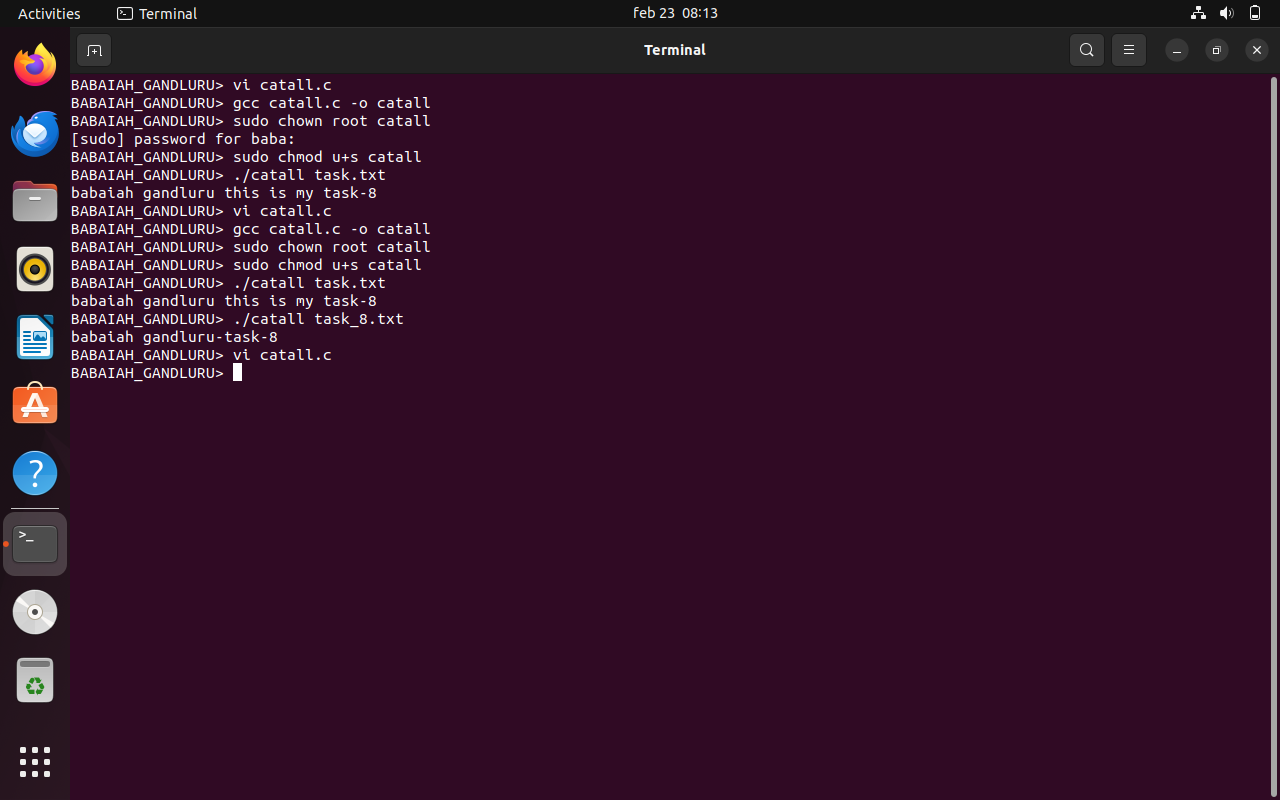
And In case If LD\_PRELOAD is set again within the program it overrides the security restriction, and the overridden sleep() function from mylib.so is called. so

I got "i am not sleeping !" message

Similarly when I executed myprog as a Set-UID user1 program which is baba in my case where LD\_PRELOAD set by a different user is not considered unless explicitly set again within the program.

So,I got "i am not sleeping !" message

***Task-8***



I am able to access the text file in the both scenarios of using the below commands

system(command);

execve(v[0], v, NULL);

I tried to acess the file task.txt and I am getting the text from task.txt file while using any one of the commands from

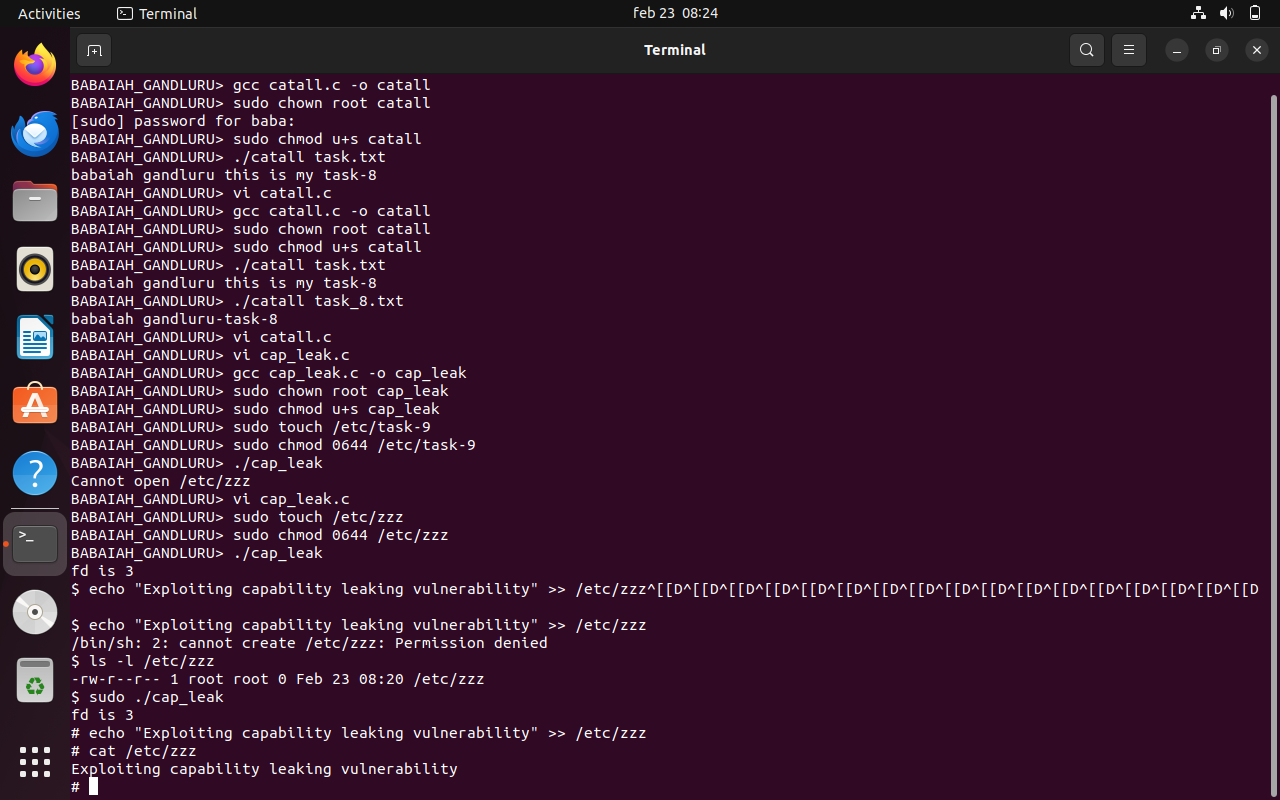
system(command);

I got acess to text file because system() invokes a shell and it allowed me to execute commands as if they were run directly from the command line with my own privileges.

execve(v[0], v, NULL);

I am still able to access the text inside the text file is that the program is directly executing /bin/cat using execve(). eventhough execve() doesn't invoke a shell. it still allows the execution of /bin/cat, which has permissions to read the contents of files. This program directly calls /bin/cat using execve() which bypasess any shell interpretation.

***Task-9***



I execude the code and sucessfully explited the vulnarbility using the code and I am able to acess the text file which i shouldnt have acess to. In this case file zzz.txt

This is because process retains some privileged capabilities even after dropping root privileges. I am able to exploit this to write to /etc/zzz as a normal user.