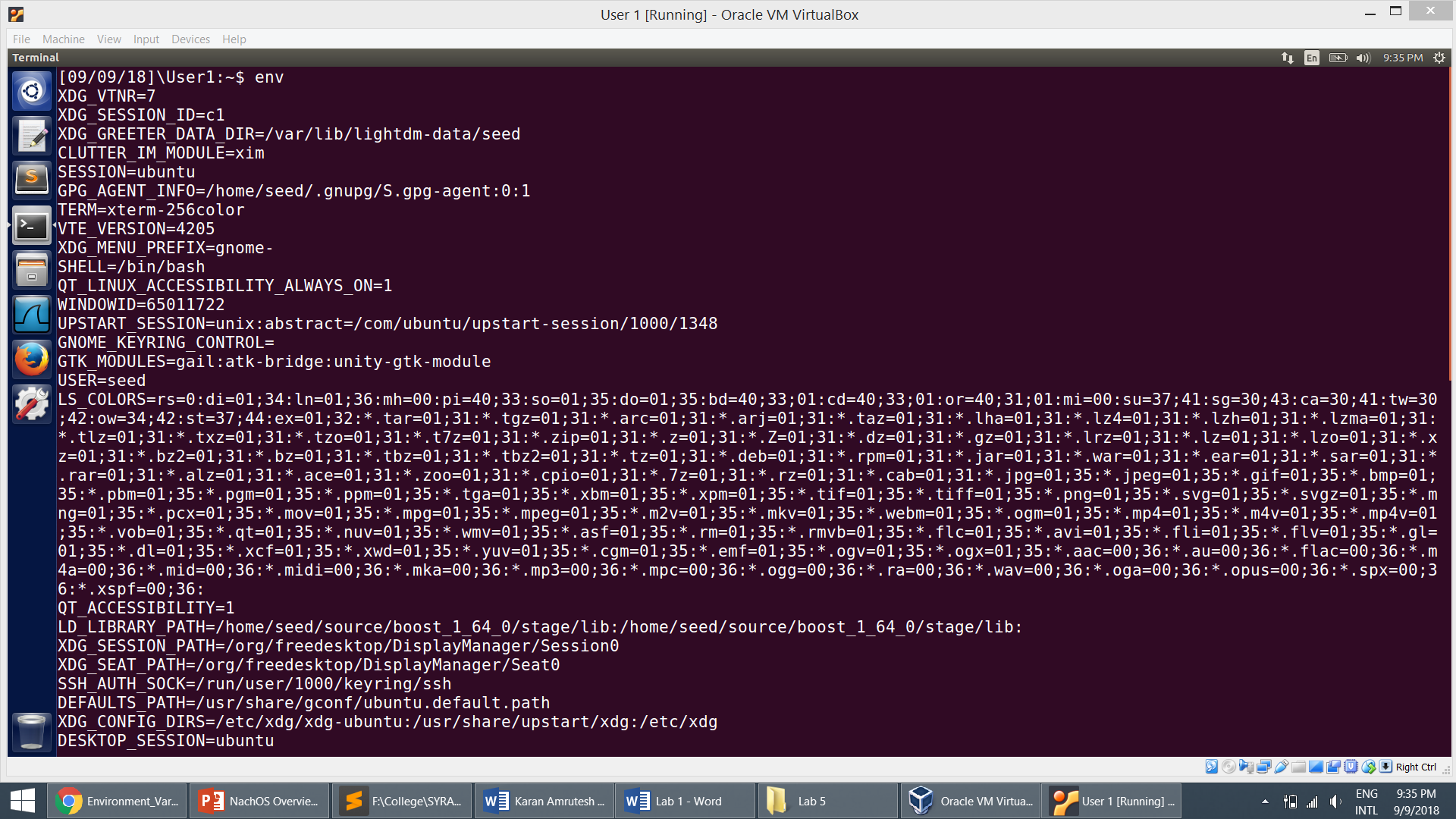
**Lab 1: Environment Variable and Set-UID Program Lab**

**Karan Amrutesh**

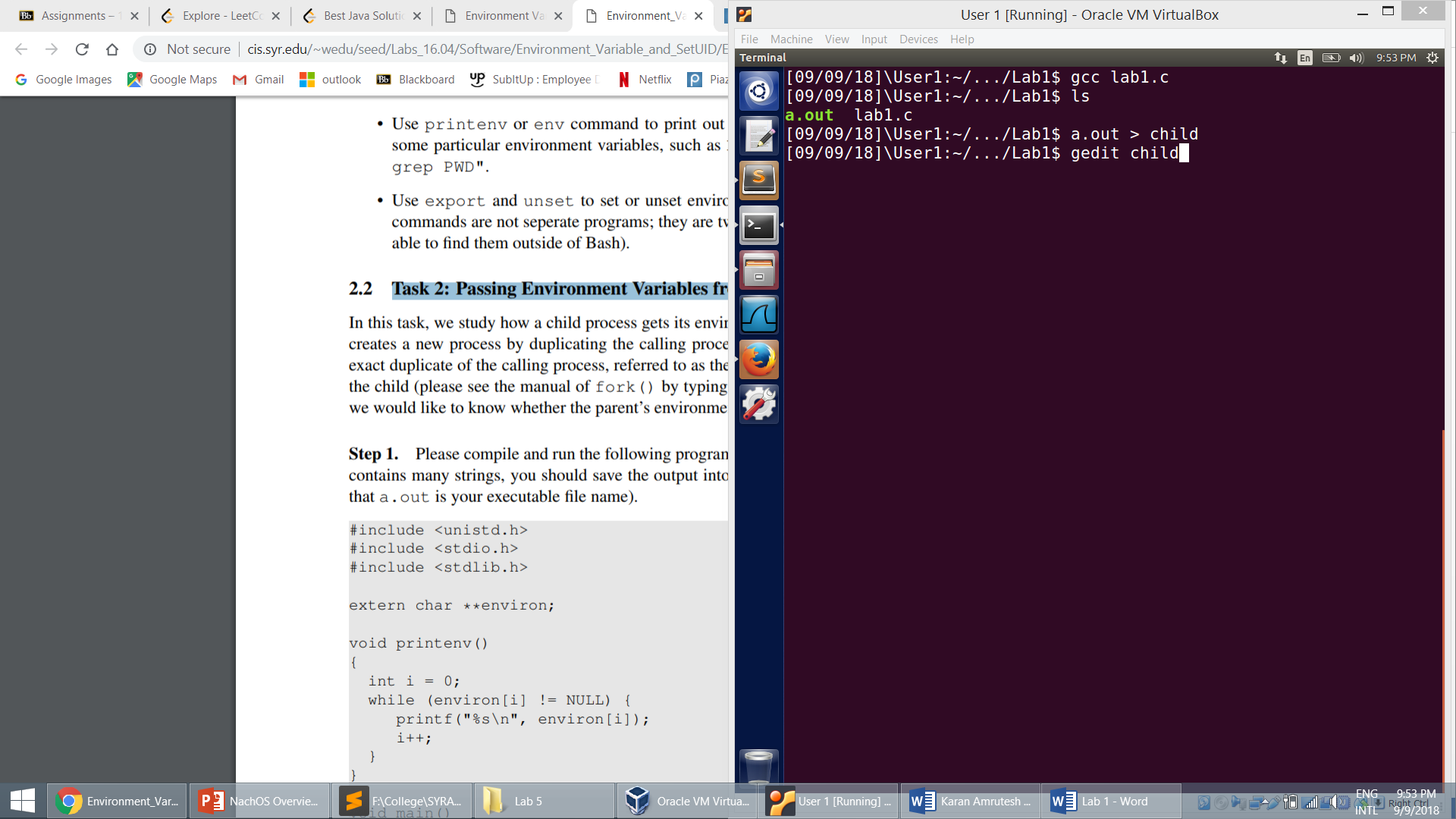
**Task 1: Manipulating Environment Variables**

* Use printenv or env command to print out the environment variables:

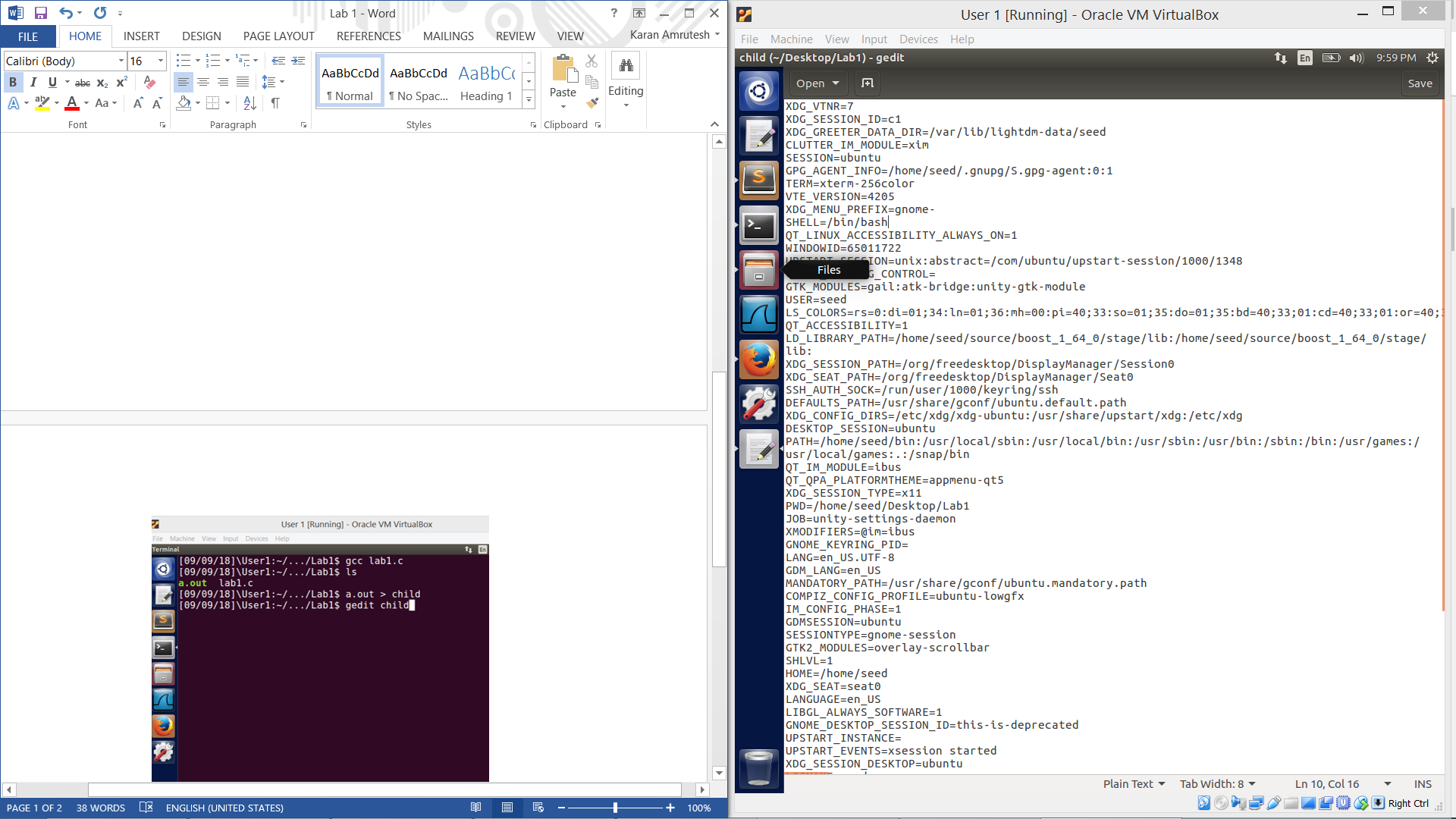


**Task 2: Passing Environment Variables from Parent Process to Child Process**

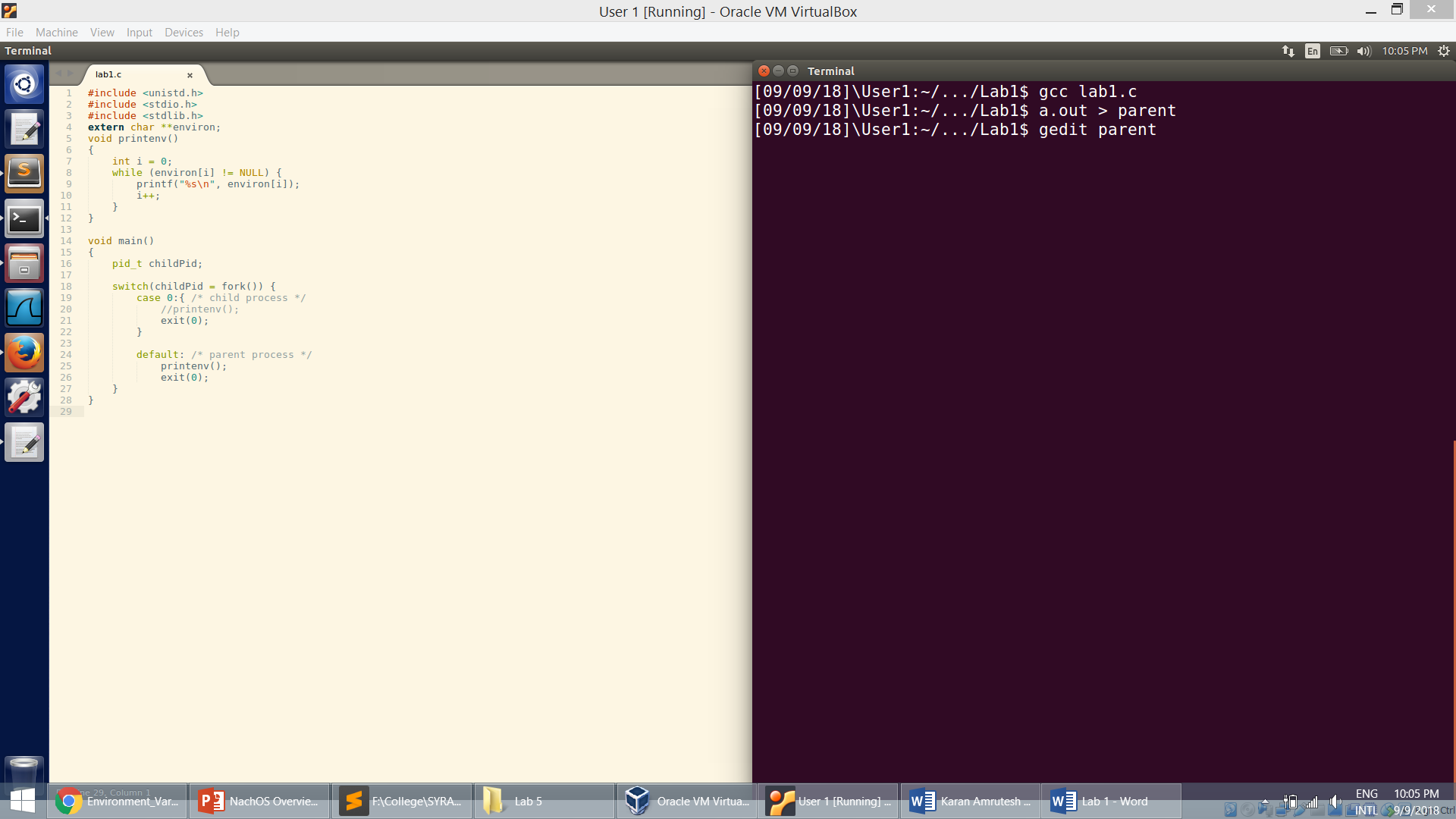
* Compiling the code copying the output to the file:



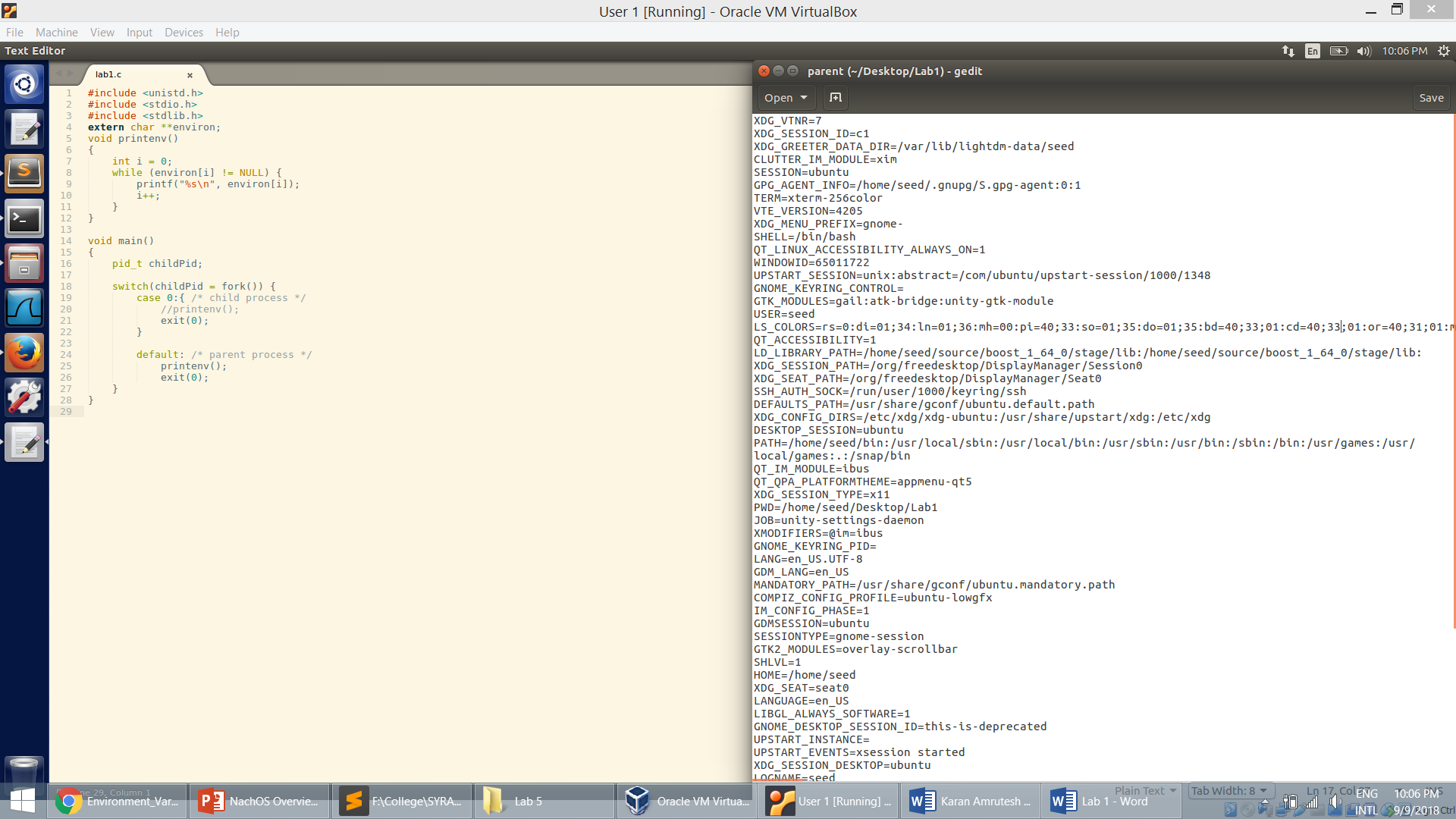
* The output file showing environment variables of the child process. We can see that the child process inherits the environment variables of the parent process.



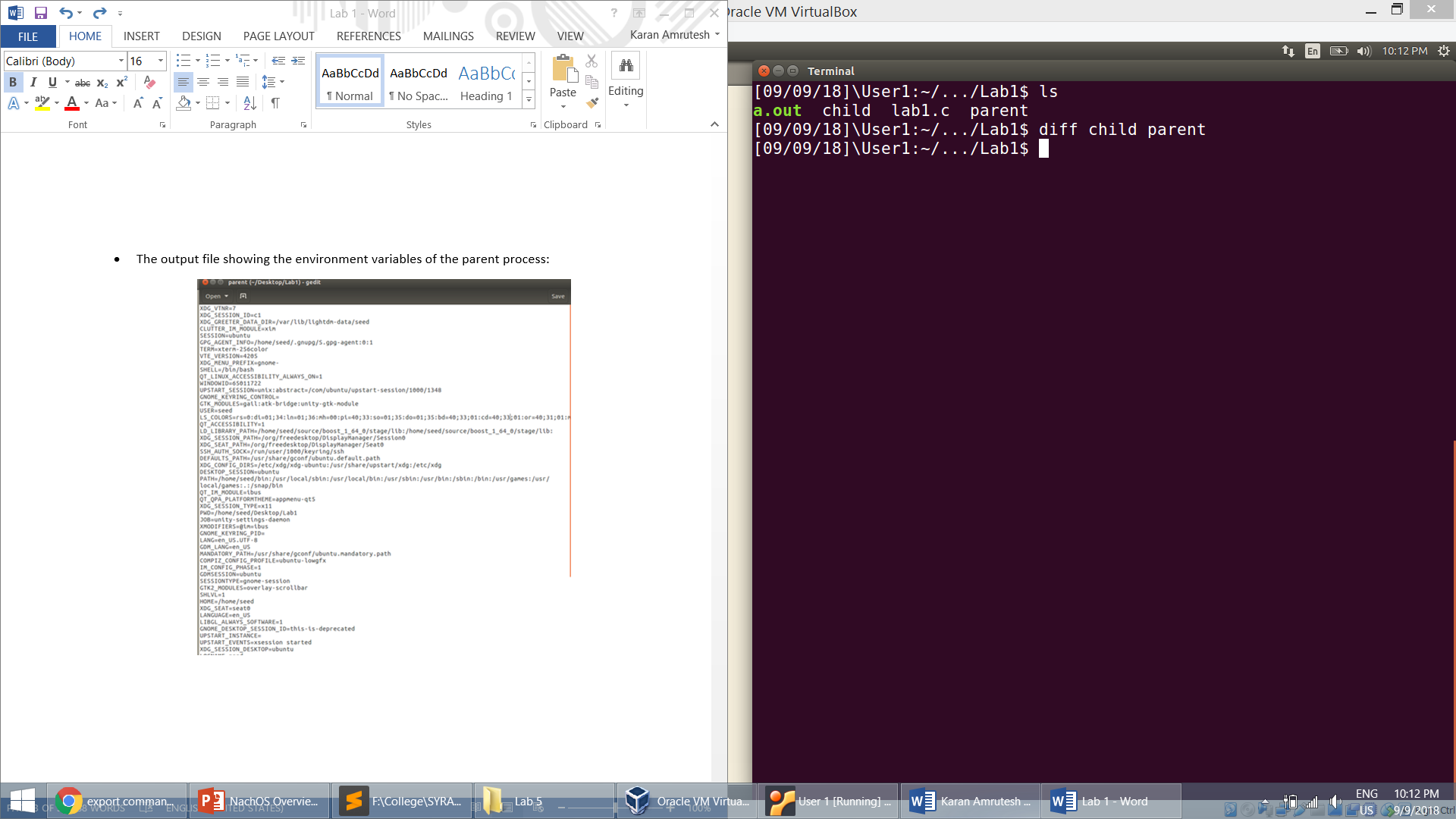
* Changing the code and recompiling:



* The output file showing the environment variables of the parent process:



* By comparing the two output files using the ***diff***command, we can see that the two files are identical and the child process inherits the environment variables from its parent process.

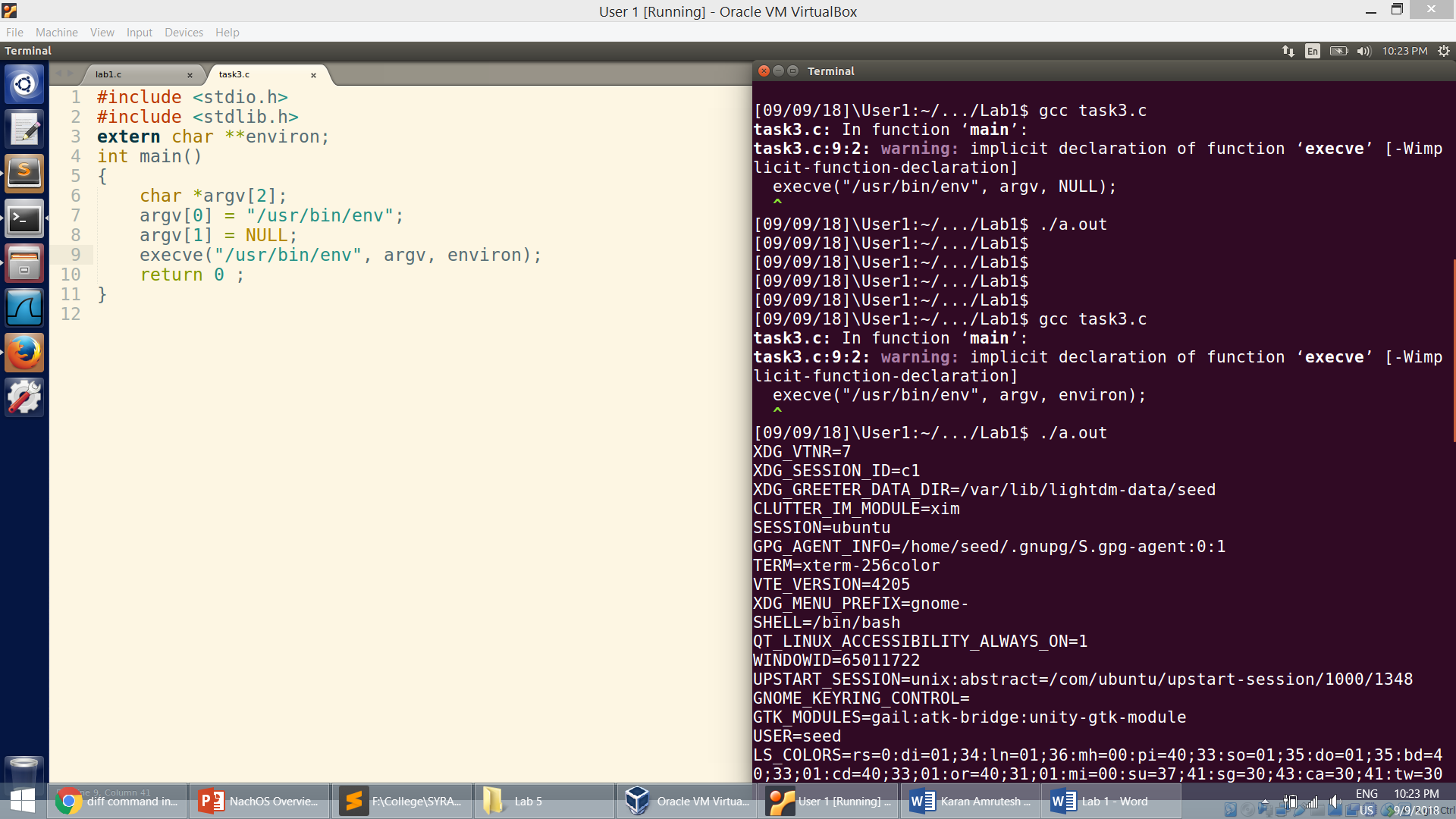


**Task 3: Environment Variables and execve():**

* Compiling the program with the NULL parameter:

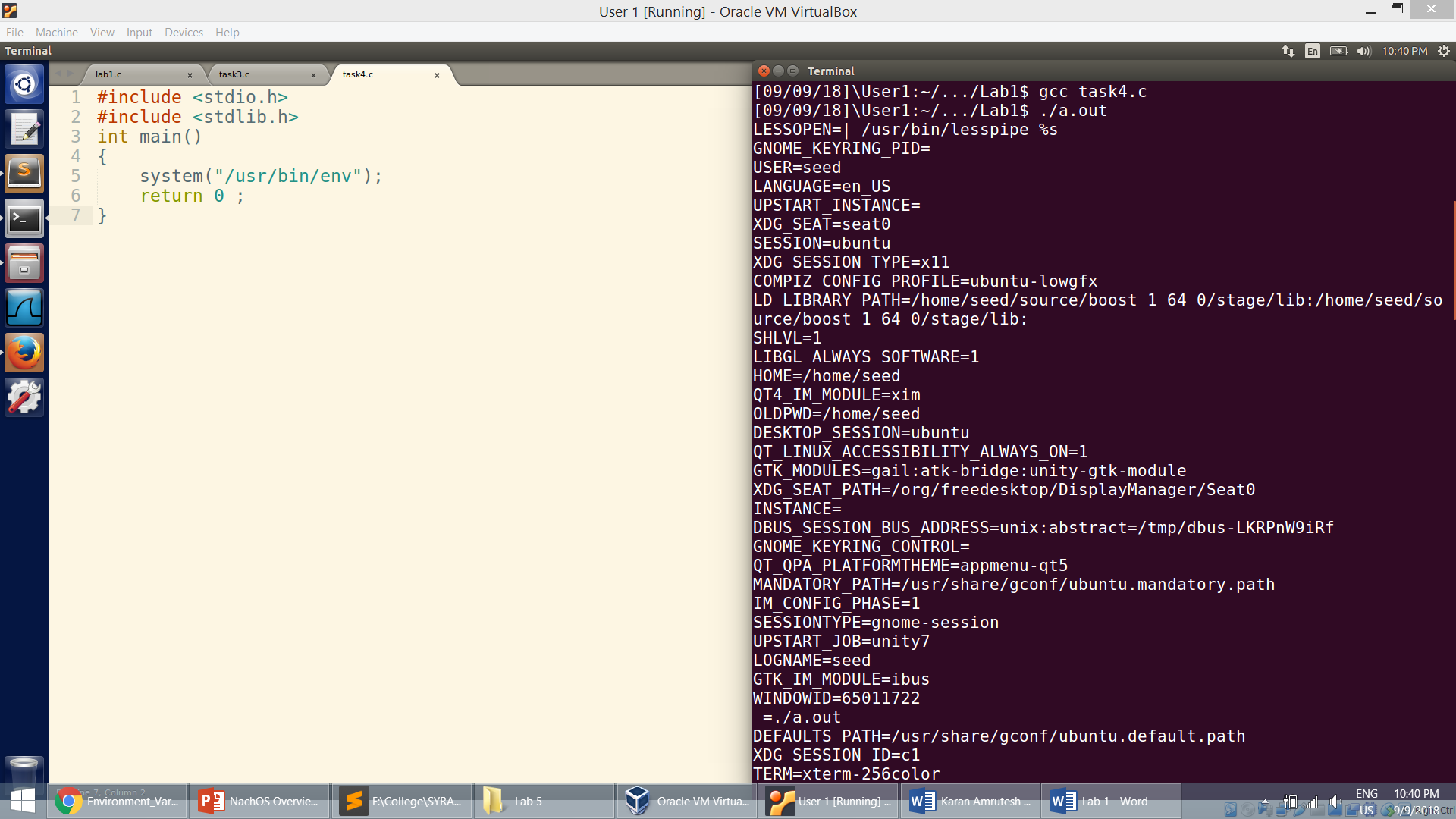


* Compiling the program with environ as 3rd parameter:



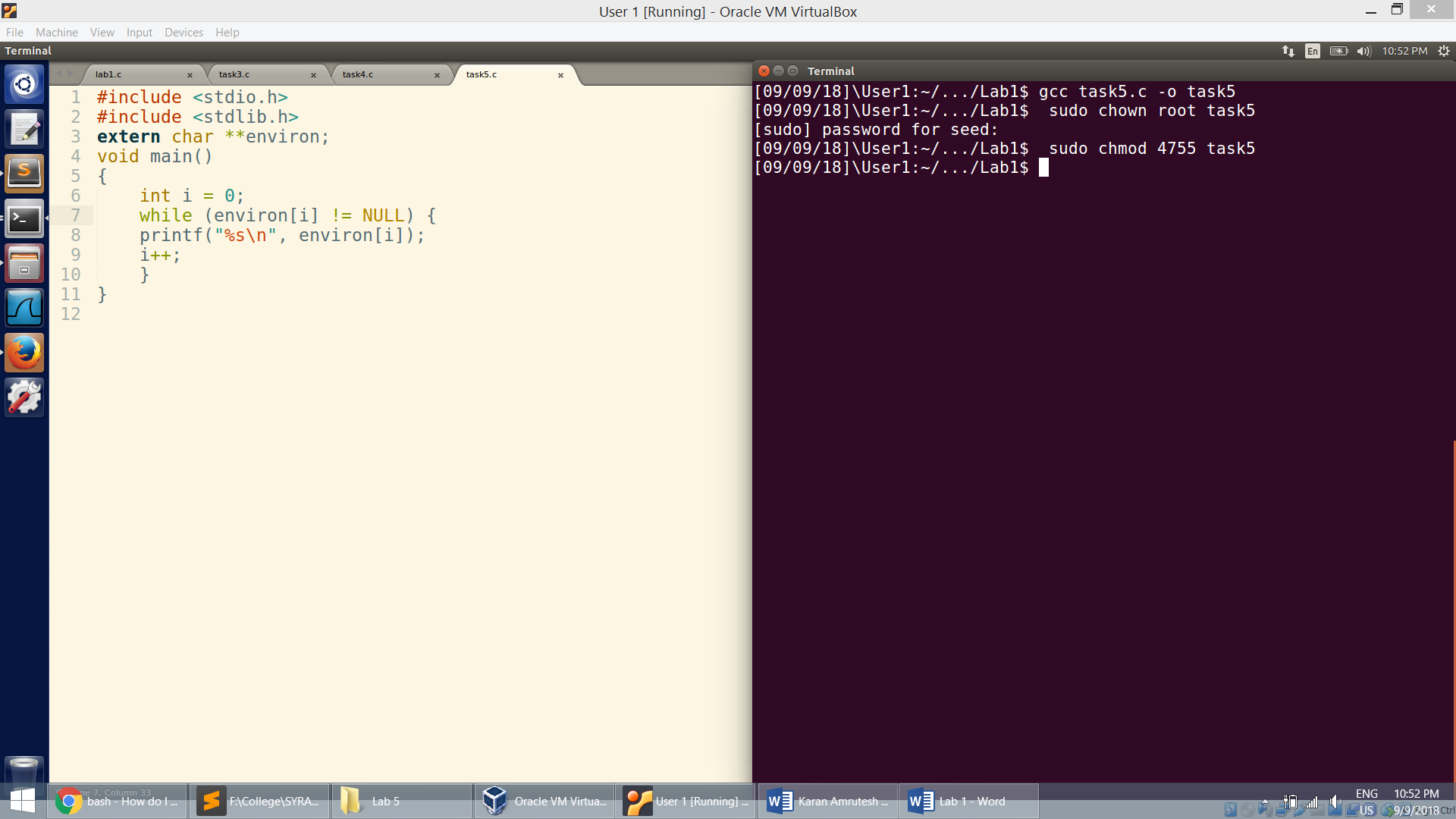
* We can conclude that the parameters for execve decides the environment variables of the process.
* The new program does not have any environment variable when we pass NULL to execve()
* By passing environ to execve, all the environment variables of the current process are passed to the new program

**Task 4: Environment Variables and system()**

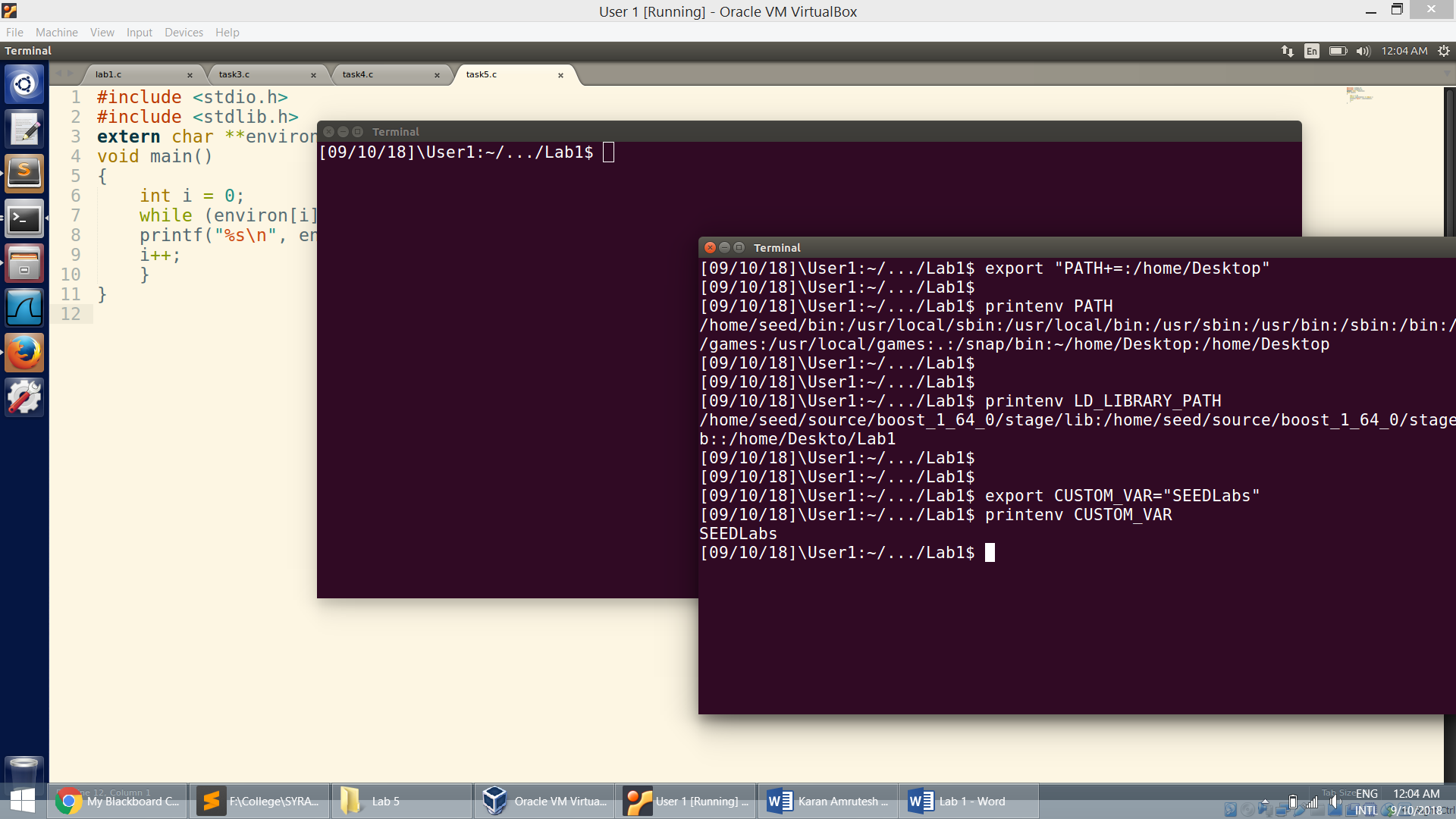


**Task 5: Environment Variable and Set-UID Programs**

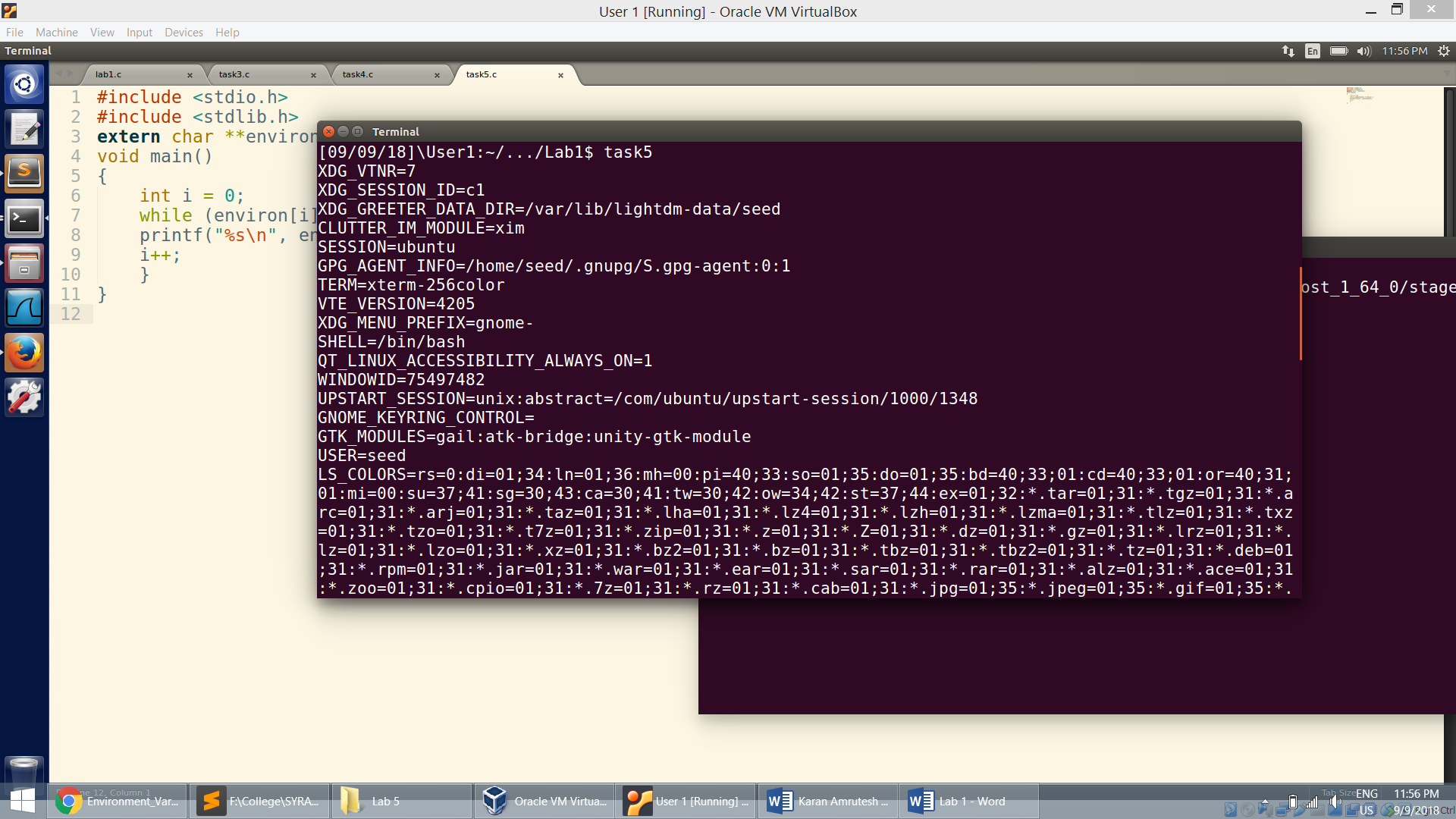
* Compiling and making it a set-UID program.



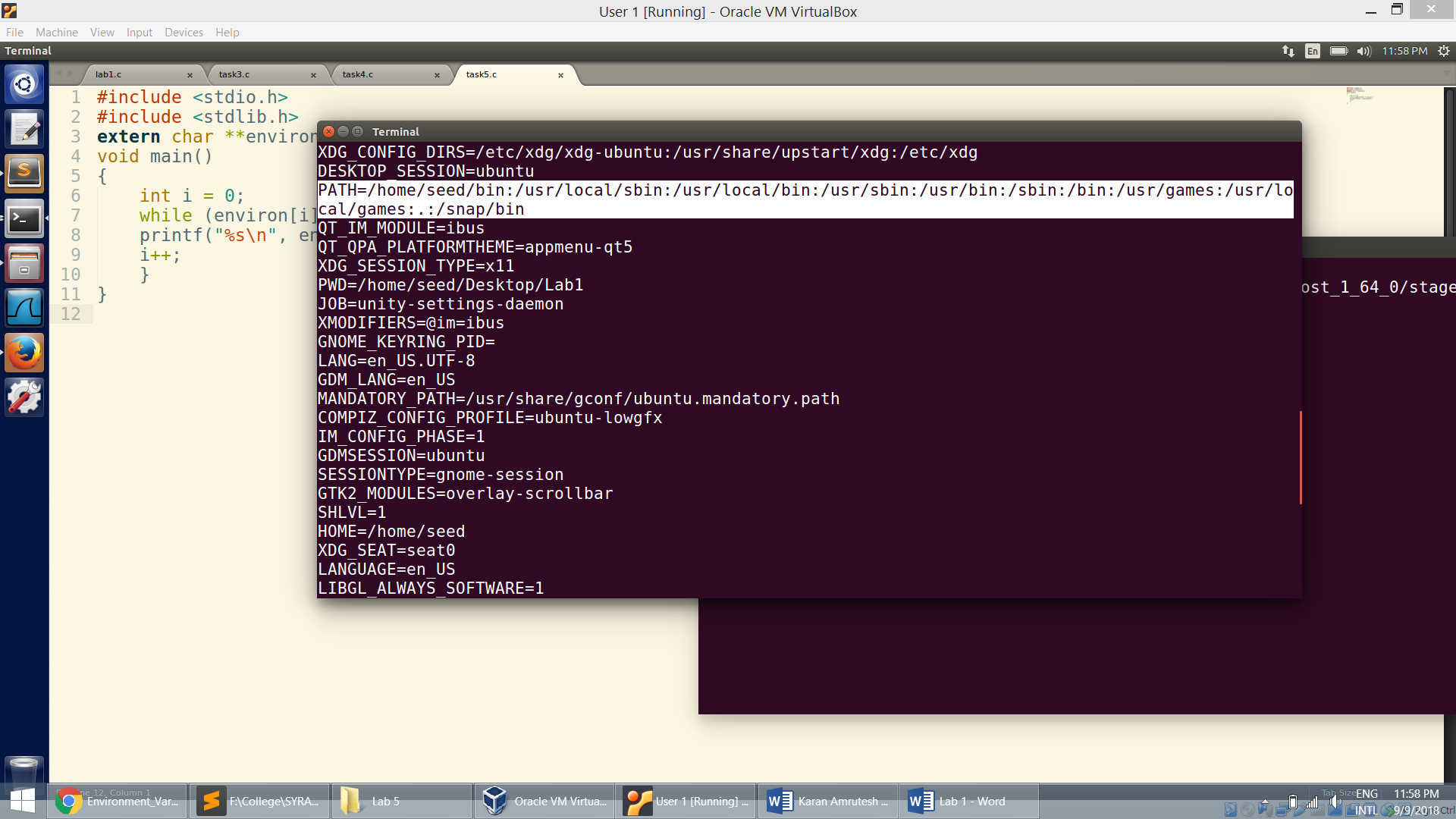
* Setting values to enironment variables in the user shell:

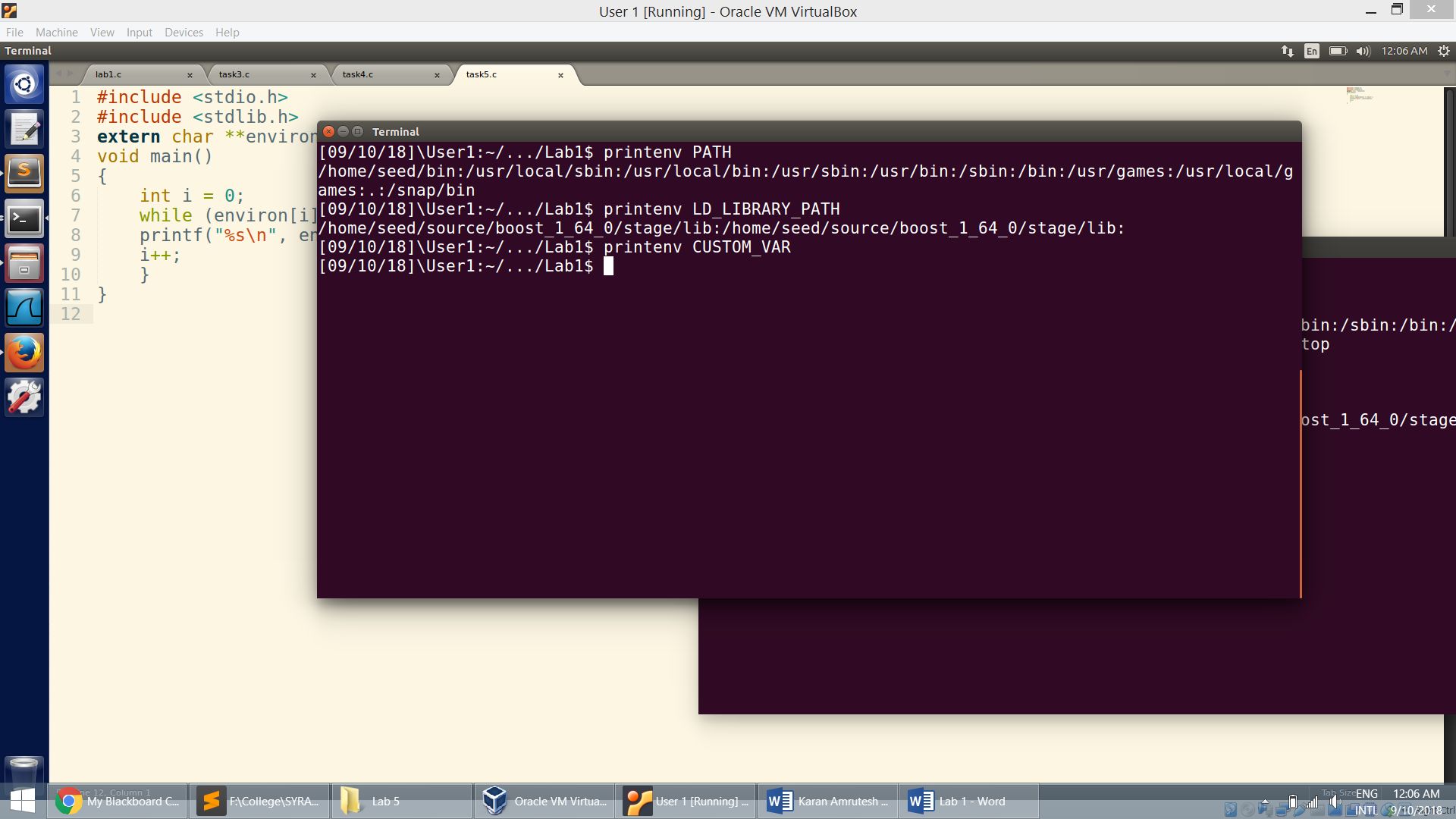


* Running the program in the shell:



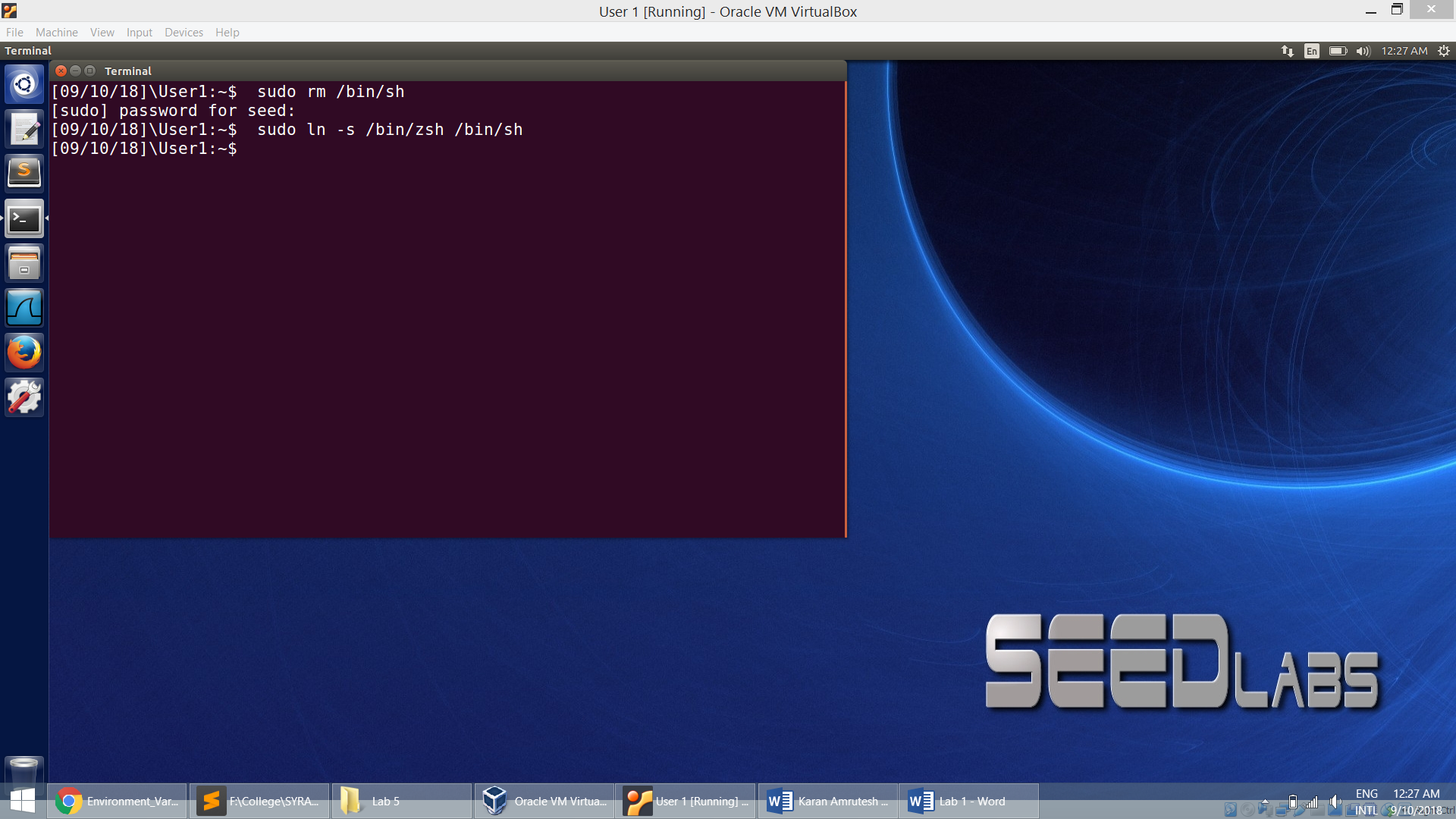
* We can see that the environment variables set in the user shell does not effect the environment variables in the root.



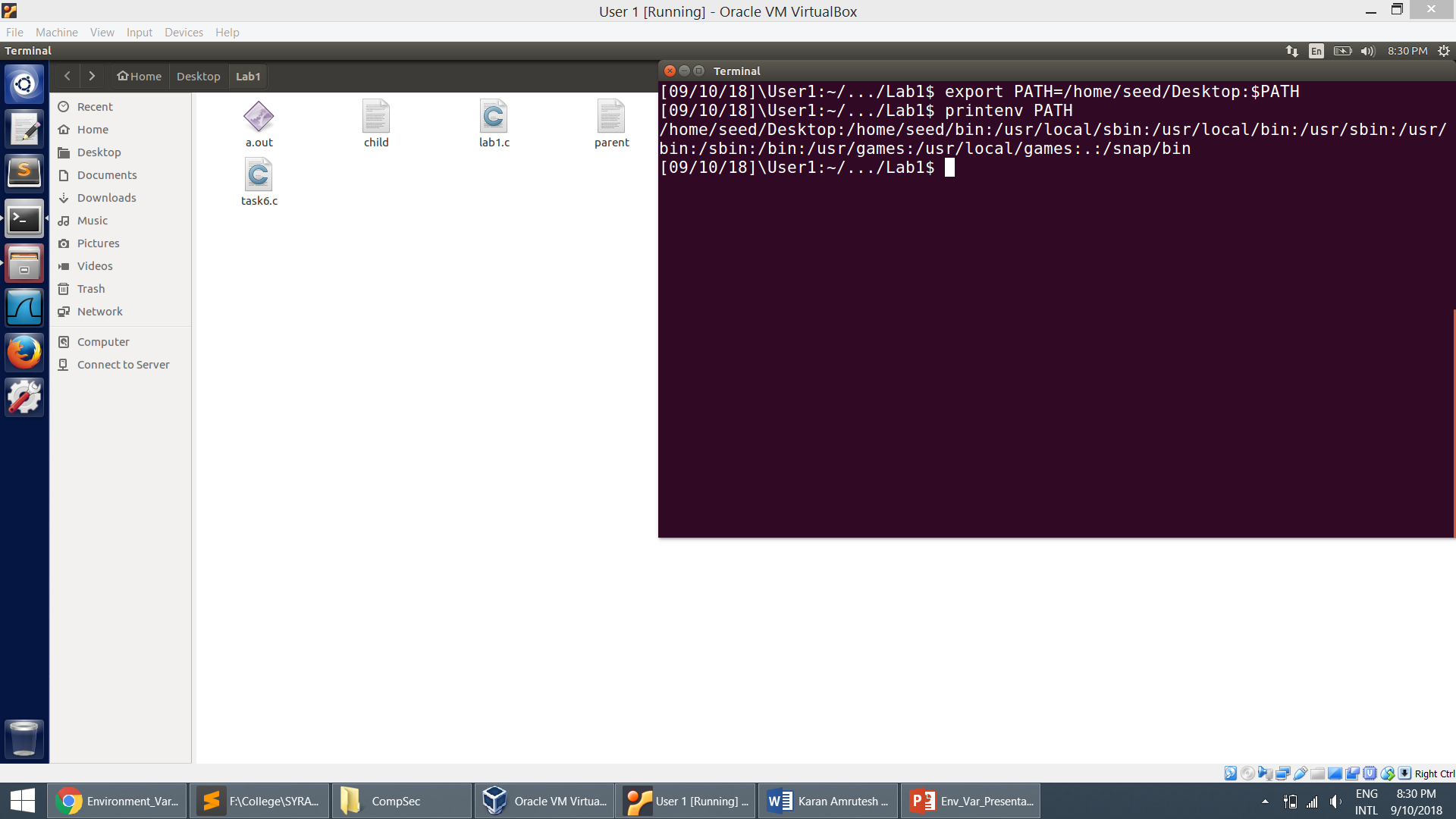


**Task 6: The PATH Environment Variable and Set-UID Programs**

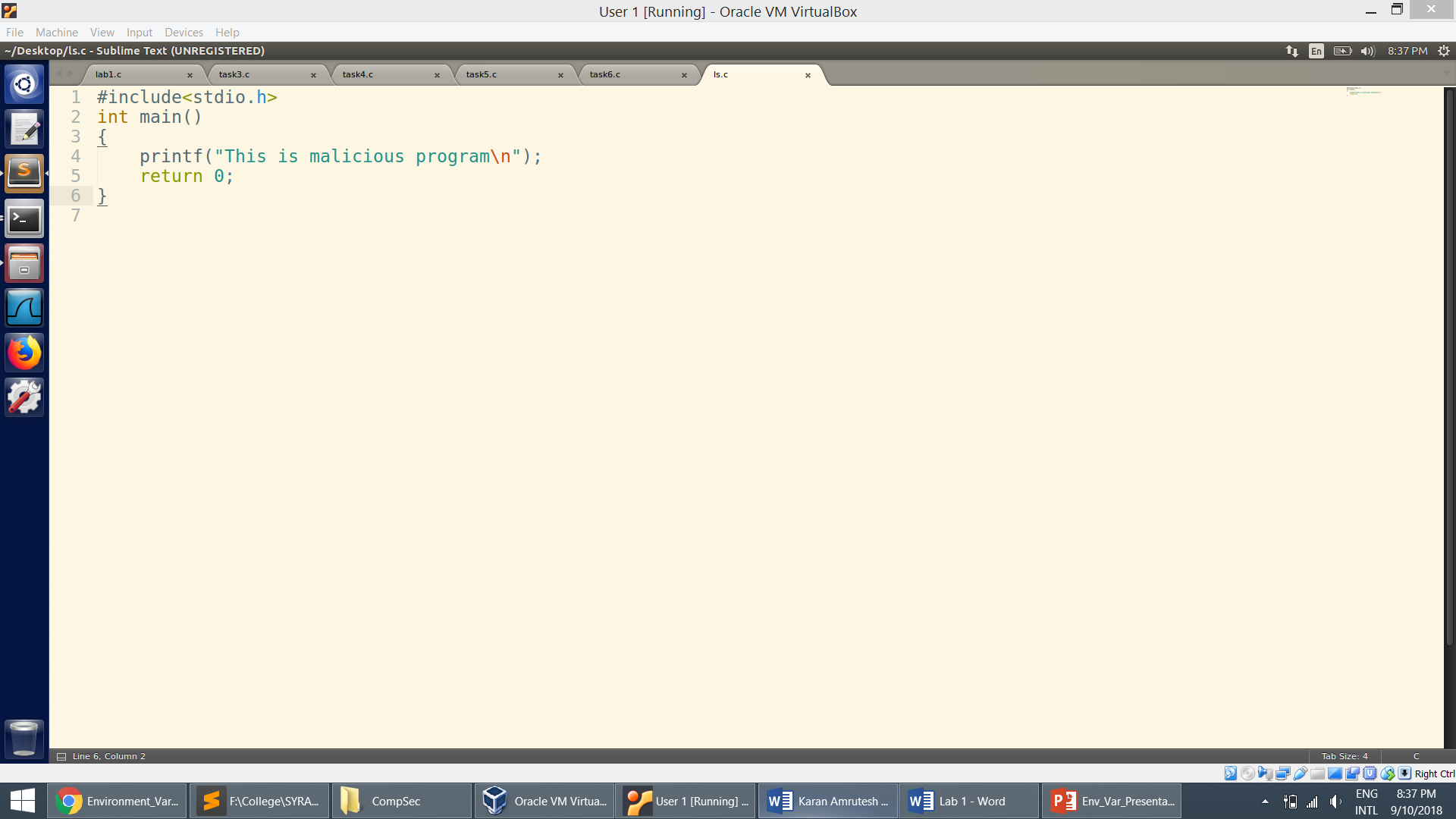
* Linking /bin/sh to zsh:



* Adding the directory /home/seed/Desktop to the beginning of the PATH environment variable:



* I place my compiled malicious ls program in the desktop:

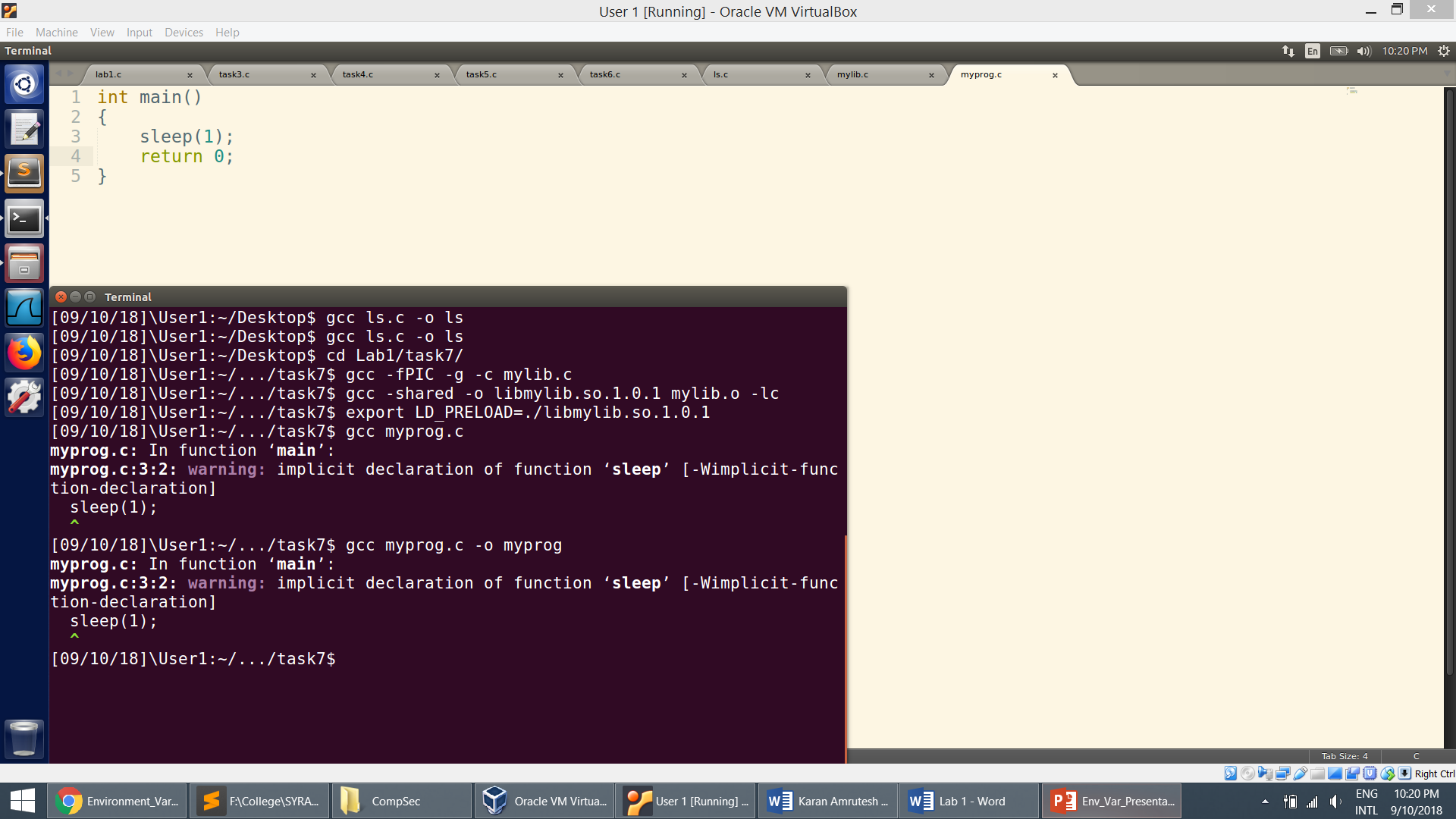


* Now when I run the setUID program which calls the system function on ls, the program checks the first directory specified in the PATH for the ls program.
* Here, it is home/seed/Desktop. It finds the malicious ls program and runs it.

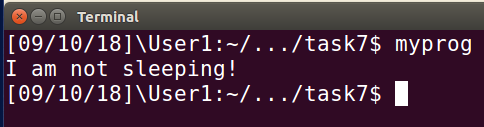


**Task 7: The LD PRELOAD Environment Variable and Set-UID Programs**

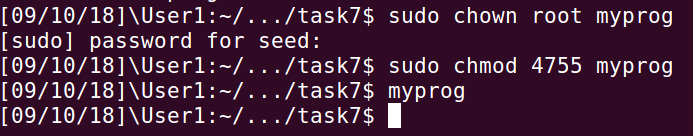
* Compiling mylib.c and myprog.c:



1. Running it as a normal user program: Our own library function gets invoked here.

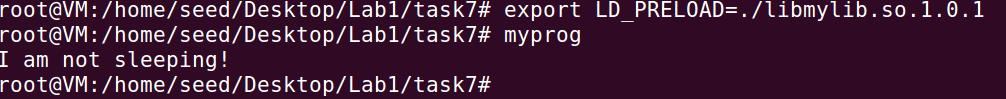


1. Running it as a set-UID root program as a normal user:



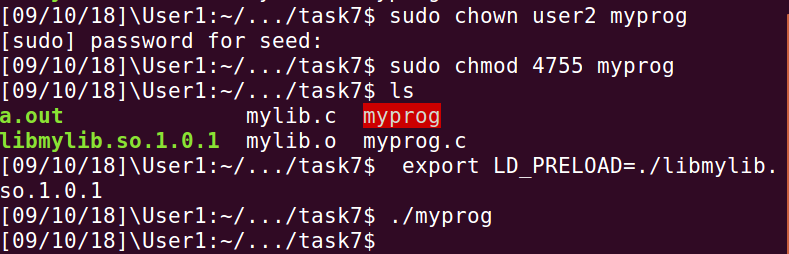
* In this situation, it will ignore LD\_PRELOAD environment variable and use the system's default sleep() function as the process’s real and effective user IDs differ.

1. Running it as a set-UID program as a root user:



* Here, it will use LD\_PRELOAD environment variable and override sleep() function.

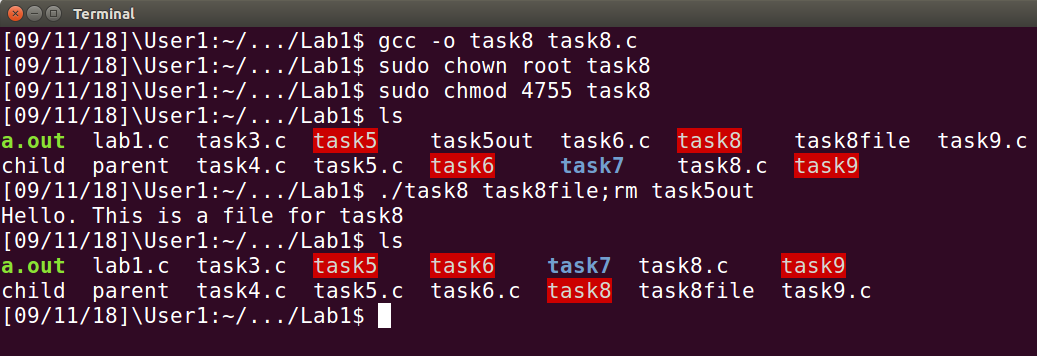
1. Make myprog a Set-UID user2 program (i.e., the owner is user2, which is another user account), and run it as a different user (User1).



* In this situation, it will not override sleep() function.

**Task 8: Invoking External Programs Using system() versus execve()**

1. Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command. 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?



* system() call invokes the shell every time it is called. So when the user is asked for a file name, he can give any other command after the semi-colon. This can compromise the integrity of the system.

1. Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID. Do your attacks in Step 1 still work? Please describe and explain your observations.



* Here we can see that using execve does not execute the rm command. This is because execve does not invoke the shell to run the command. This way it is safer the than the system call.

**Task 9: Capability Leaking**

* Since the program is a root-owned set-UID program, it will have root privileges. Though the program is not run as root, the data is still written to the file as the file descriptor, opened when having root privileges, is still available.
* Using this, the malicious user writes into the file.
* This is capability leaking.

