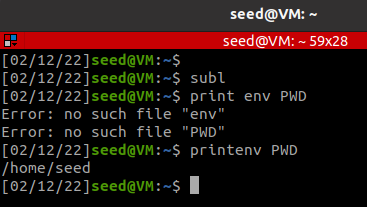
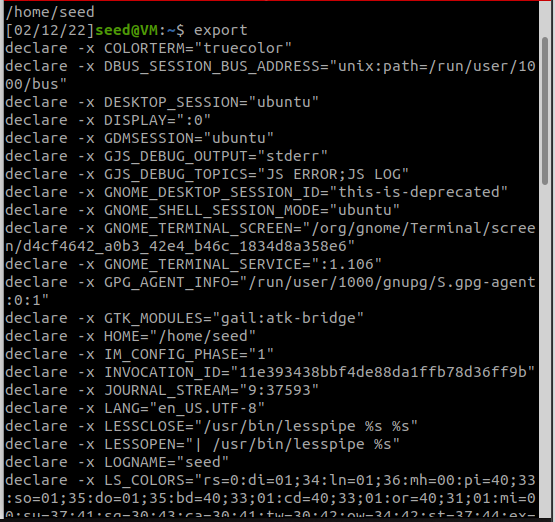
Val Robichaux

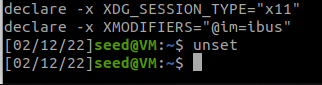
Homework 2

CSCE 465

**Task 1:**

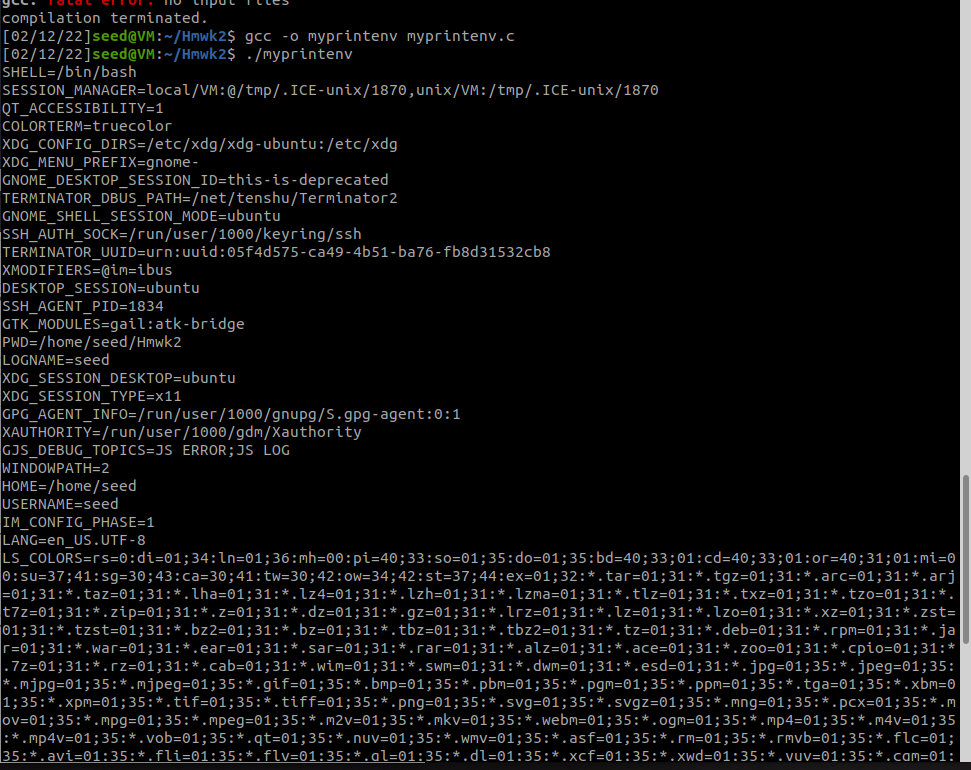




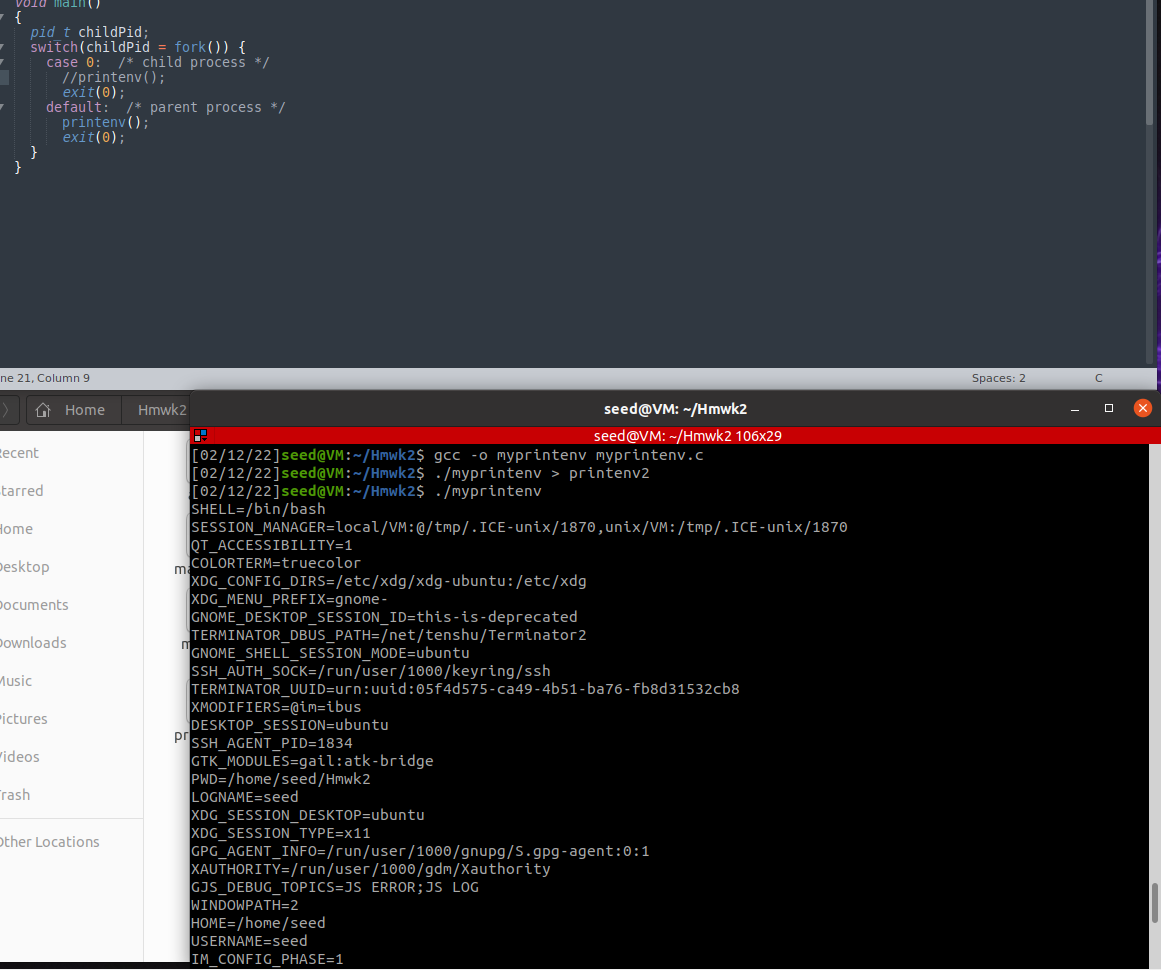


**Task 2:**

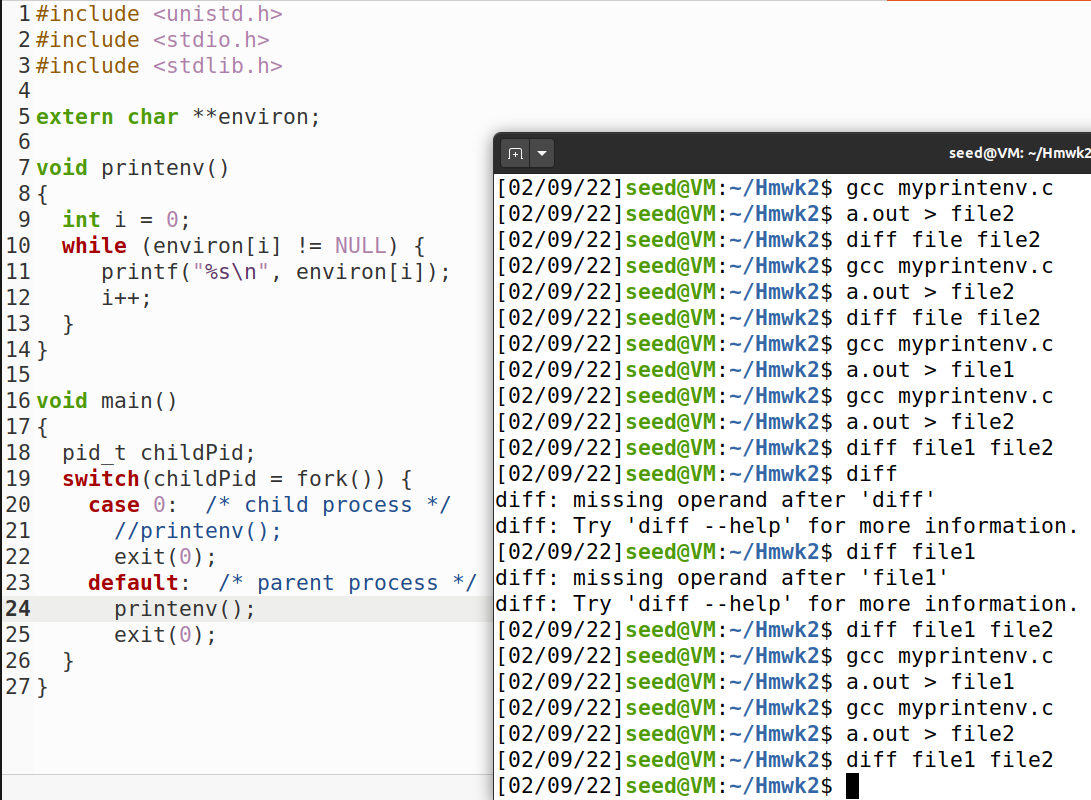
**Step 1:**

****

* Prints out my environment variables!

****

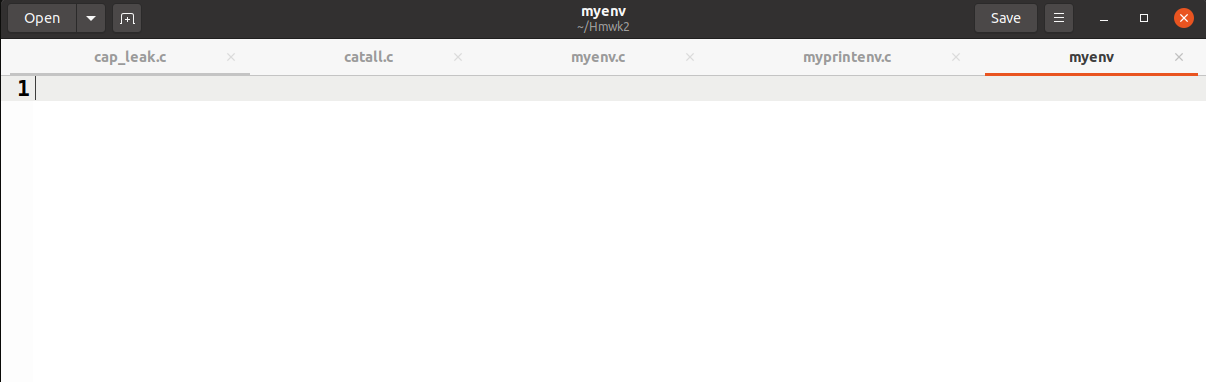
* It still prints out my environment variables even in the parent process.

****

* Here I ran the diff of these two files after compiling the two different sets and it seems as if they are identical. This seems odd as I would expect that the child process and parent process would have some differences. I have re-ran and debugged my process in this specific task as you can see, but I have come to no different conclusion other than that the files are the same.

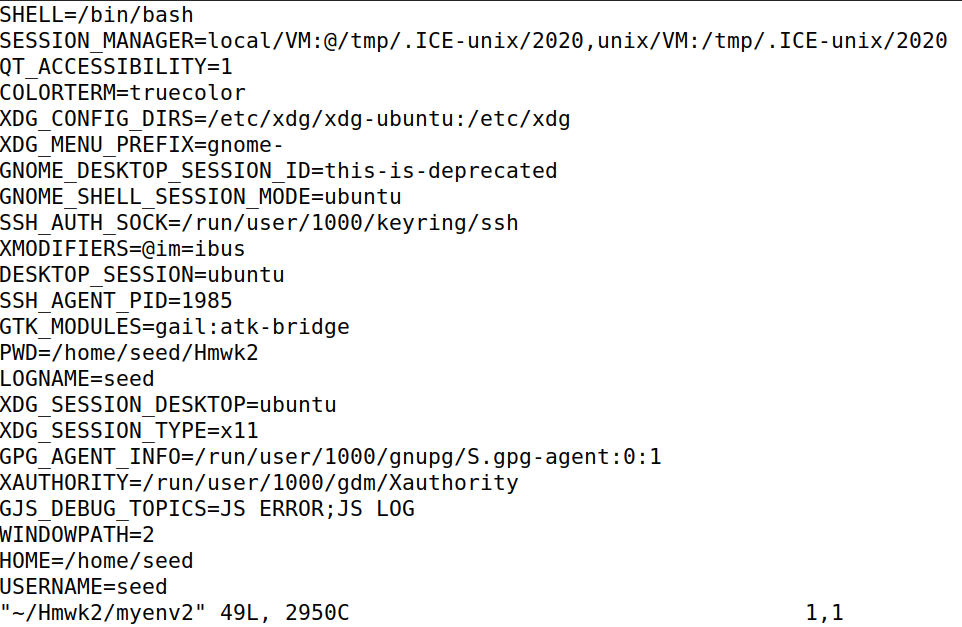
**Task 3:**

| #include <unistd.h>  extern char \*\*environ;  int main() {  char \*argv[2];   argv[0] = "/usr/bin/env";  argv[1] = NULL;   execve("/usr/bin/env", argv, NULL);    return 0 ; } |
| --- |

****

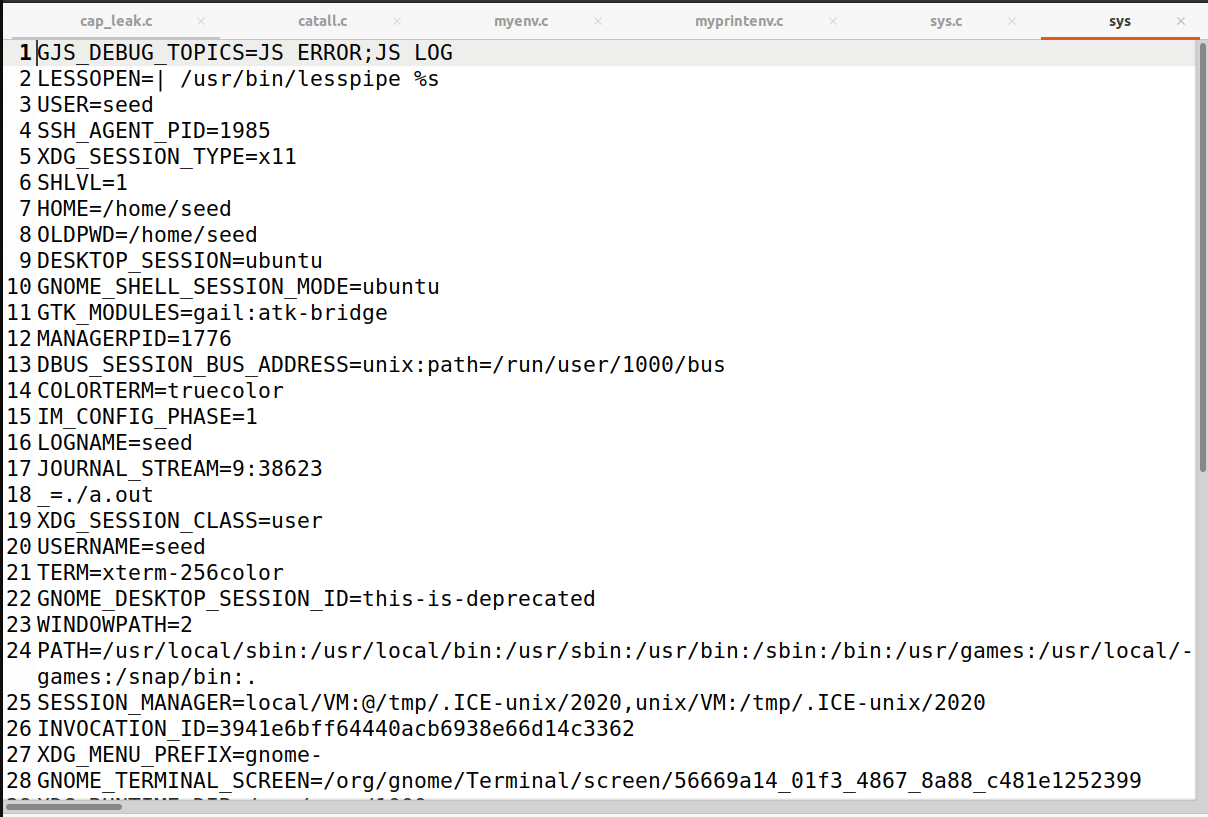
* This block of code seems to not inherit the environment variables.

| #include <unistd.h>  extern char \*\*environ;  int main() {  char \*argv[2];   argv[0] = "/usr/bin/env";  argv[1] = NULL;   execve("/usr/bin/env", argv, environ);    return 0 ; } |
| --- |



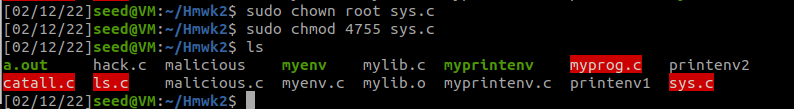
* Whilst this block of code correctly inherits the environment variables
* The execve() command executes a program referred to by the pathname.
* The first argument is the PATH
* The second argument is an array of pointers to strings that passes to the program as command-line arguments
* The third argument is an array of pointer to strings, that are passed as the environment of the new program.
* This last point is extremely important because it shows us that firstly the environment and all of its inherited variables are NULL
* In the second example it is easy to see that we are passing our local environment to the function and thus getting our local environment variables returned to us.
* It is confirmed that environment variables are **NOT** automatically inherited

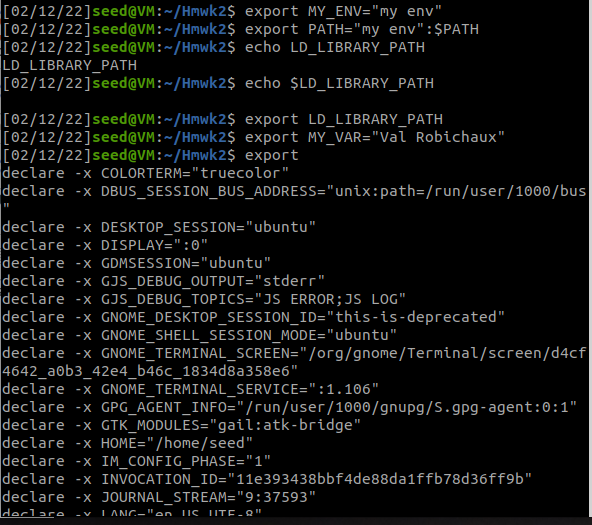
**Task 4:**

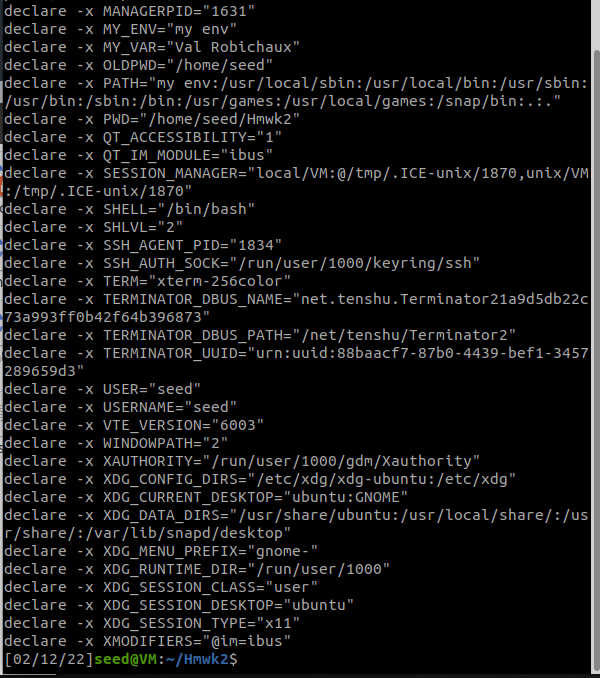
****

* Verified

**Task 5:**

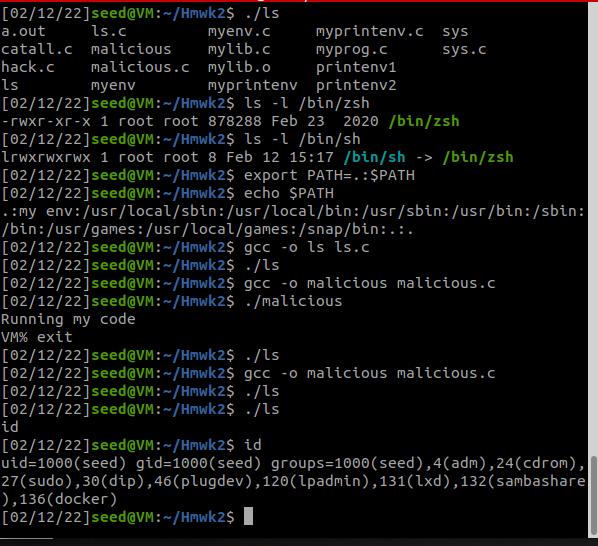
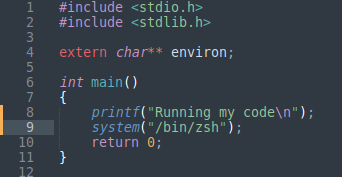
****





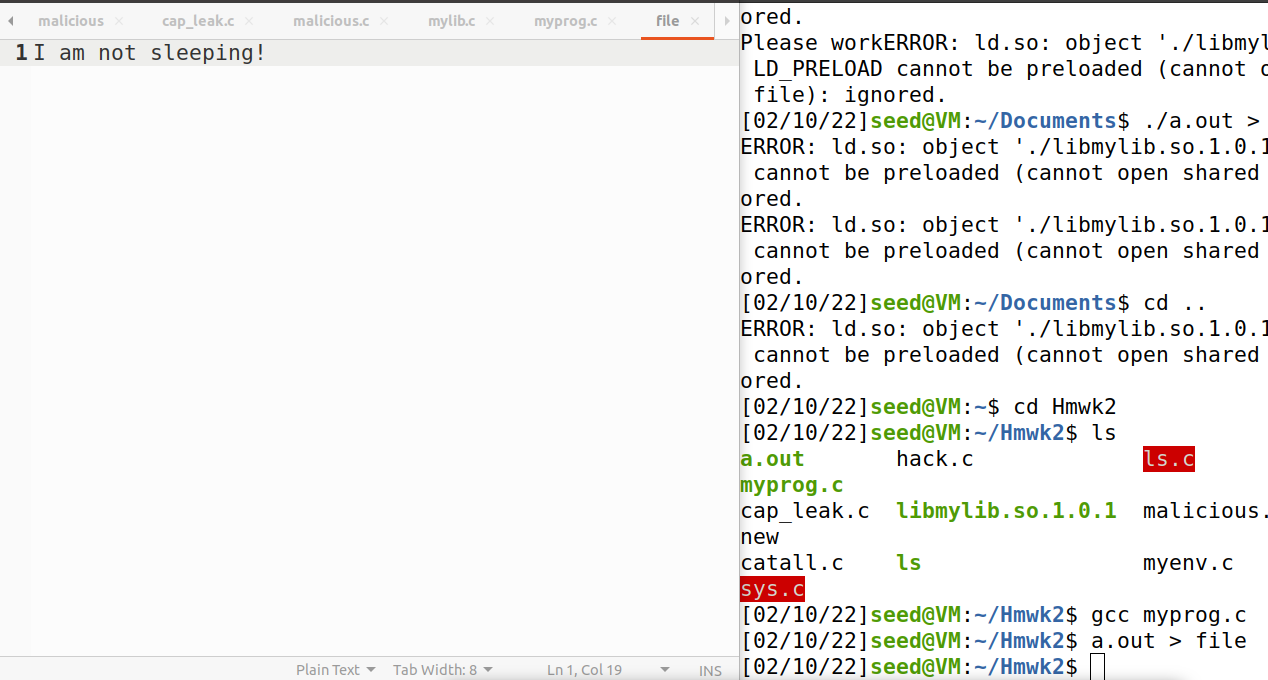
* It worked! As you can see I export my MY\_VAR variable to my own name and it shows up in the environment variables on my set-UID program. That’s really neat.
* My PATH variable seemed really surprising because it seems to point to multiple different locations including some that I am surprised with such as bin:/usr/games

**Task 6**

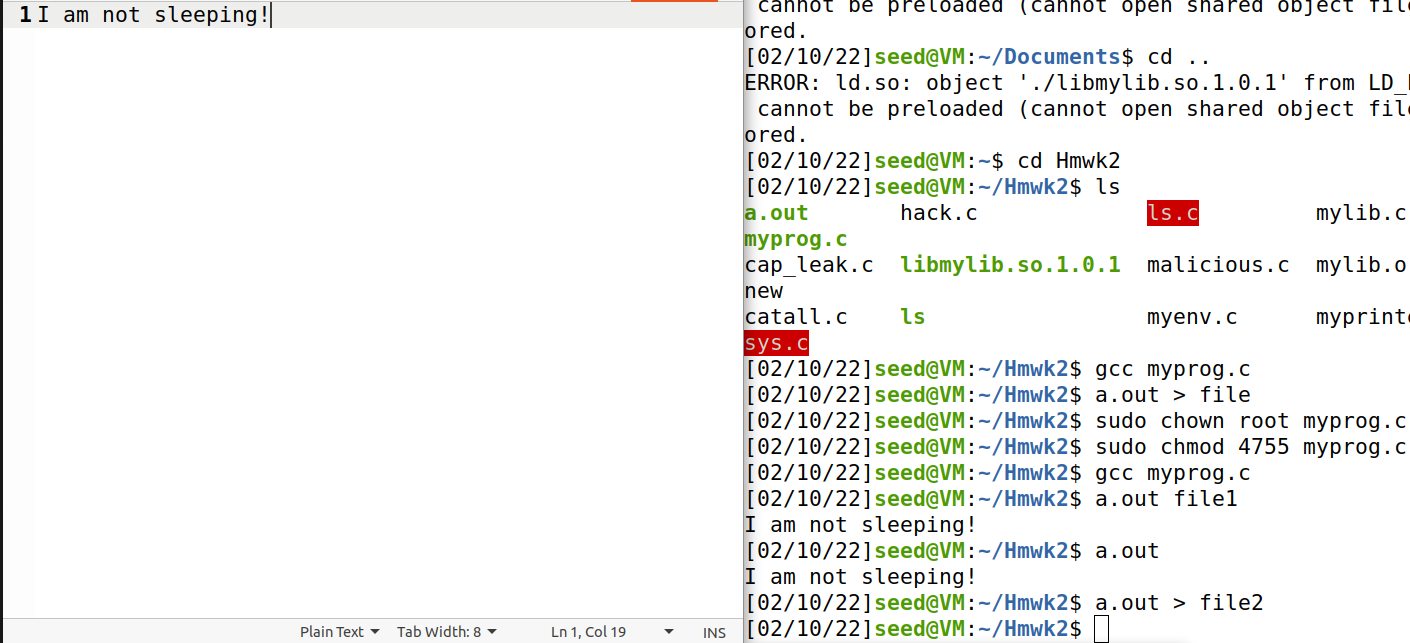


* I am having a hard time running my own code, but I can still make the original code run something else! I changed the path so that I could possibly get it to run by the absolute path instead of the relative path but could not navigate to run my malicious code
* It also seems as if the malicious code that I am running does not have root privileges as you can see from the ID call I made.

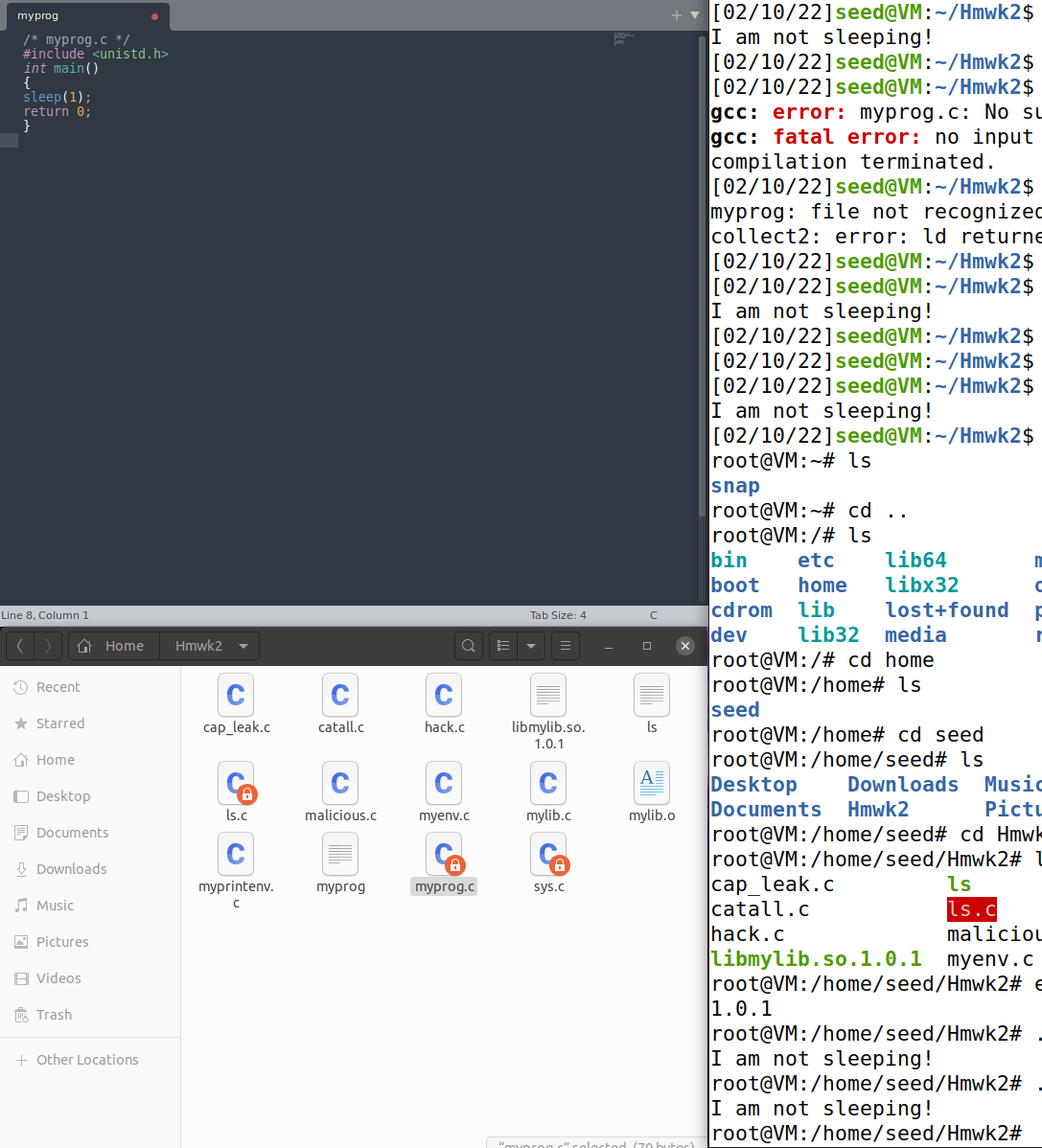
**Task 7**



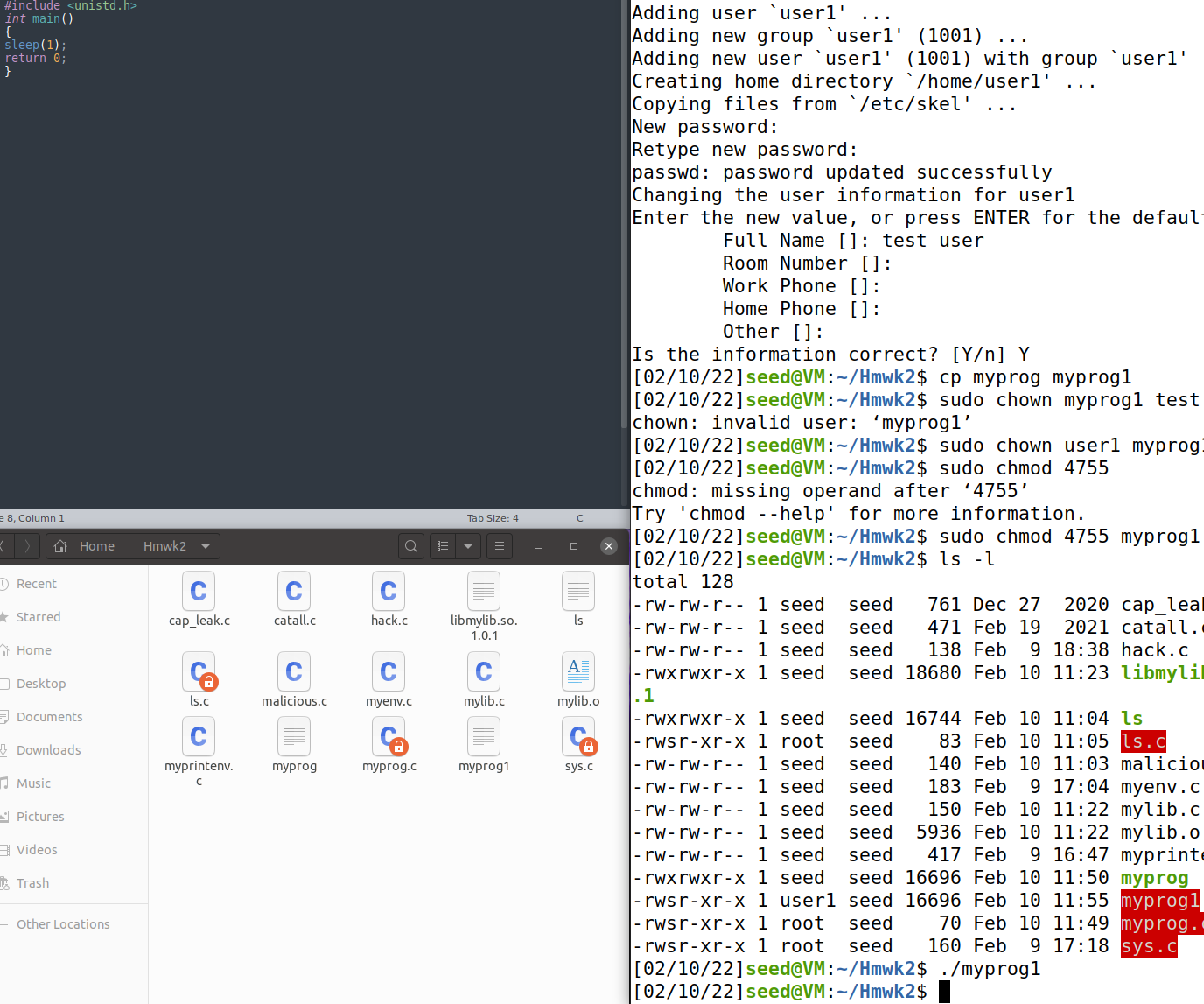
* My prog running as a normal file



* My prog running as a SET-UID program



* Ran as a root use and changed LD\_PRELOAD

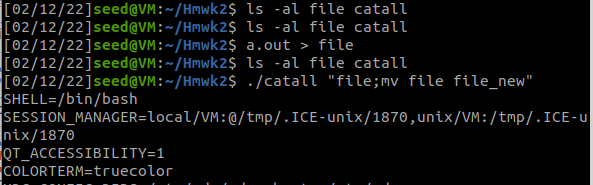


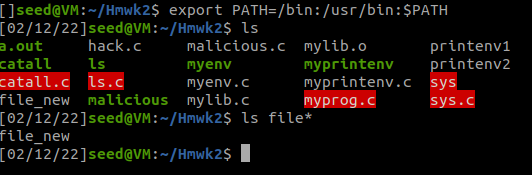
* Ran as user1

**Conclusion**

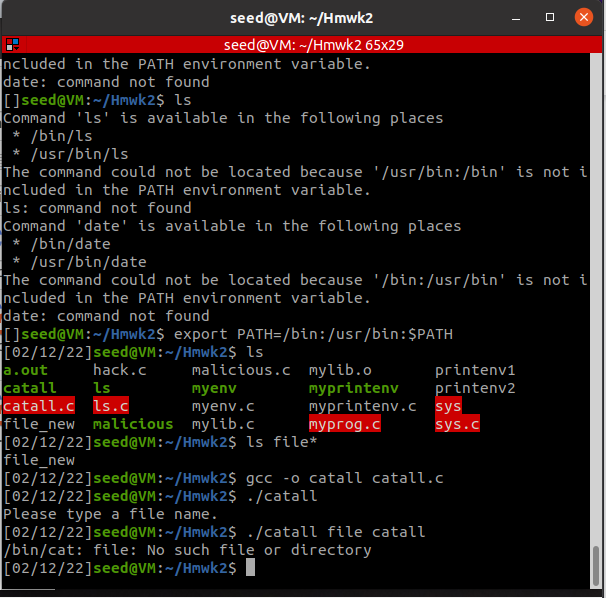
When the program is run either as a normal program of a SET-UID program, by exporting LD\_PRELOAD, the program invokes the sleep function in **libmylib**. However, this is not the case when the function is run by user1. It will always call the sleep function in the default libraries. The LD\_preload environment variables are not inherited.

**Task 8**



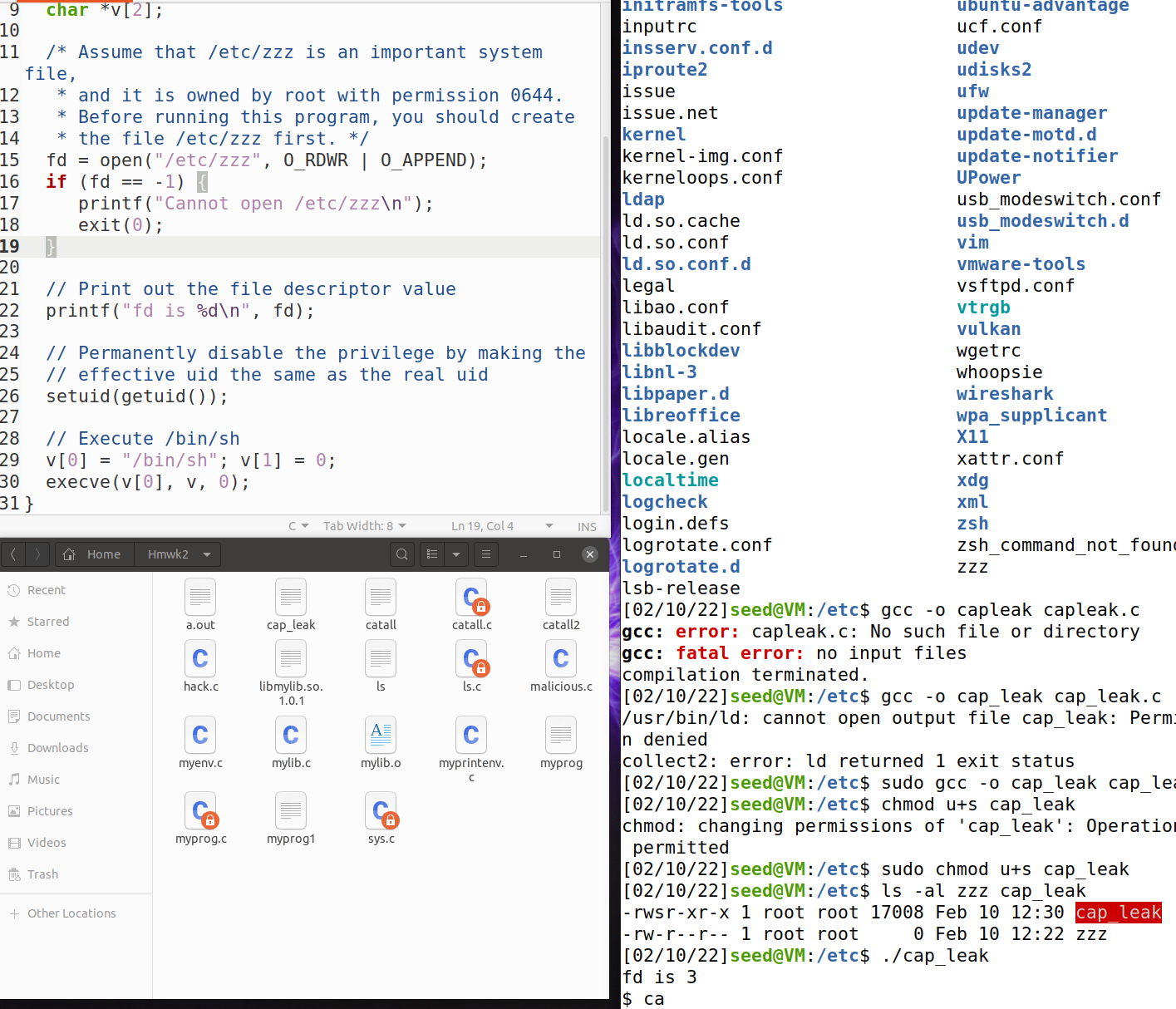


* Bob can remove, move, write to any file.
* We can change to the root user and compromise it by invoking a new shell and modifying the file.



* The attack that I used will not work because the system() call /bin/sh links zsh. Ater running the program with root privileges I can write and move files freely. When running with execve() it will see my attacks as a folder name, so the system will have no prompt or usage of the file.
* It can’t even find my file in this case

**Task 9**



The contents of the file have been modified and the reason is that file zzz is opened before the UID is set. It could be fixed if we simply moved setuid(getuid()) in front of the open() function.