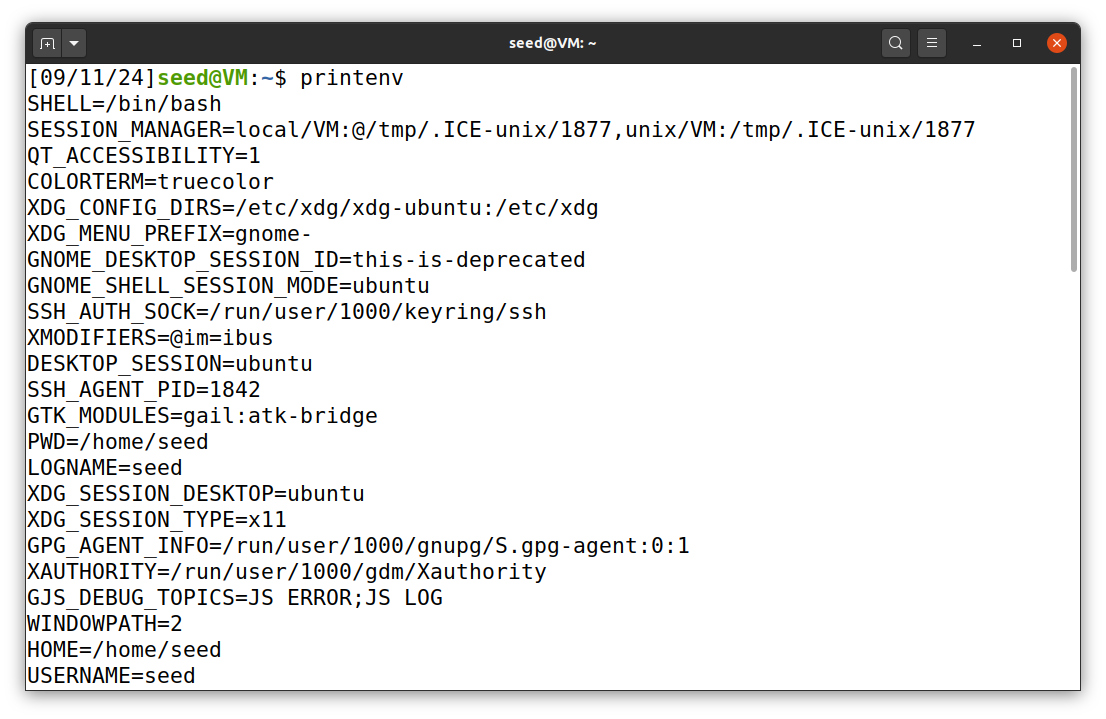
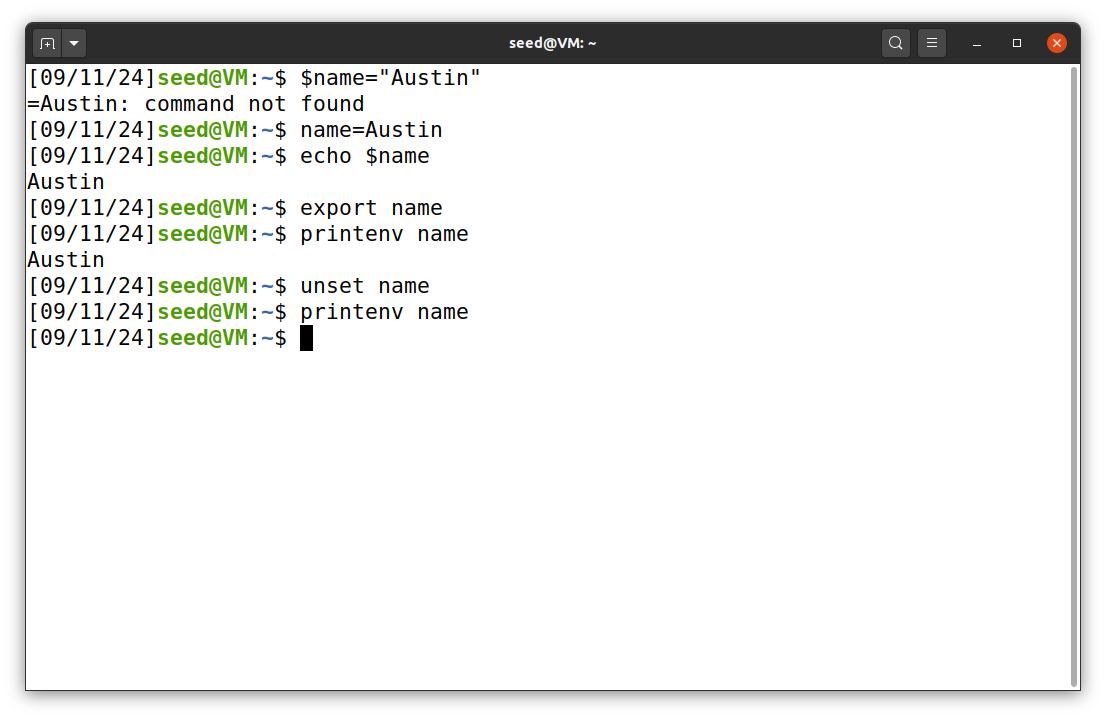
Austin Decker  
CS 445 - Computer Security  
20 September 2024

# Lab 02

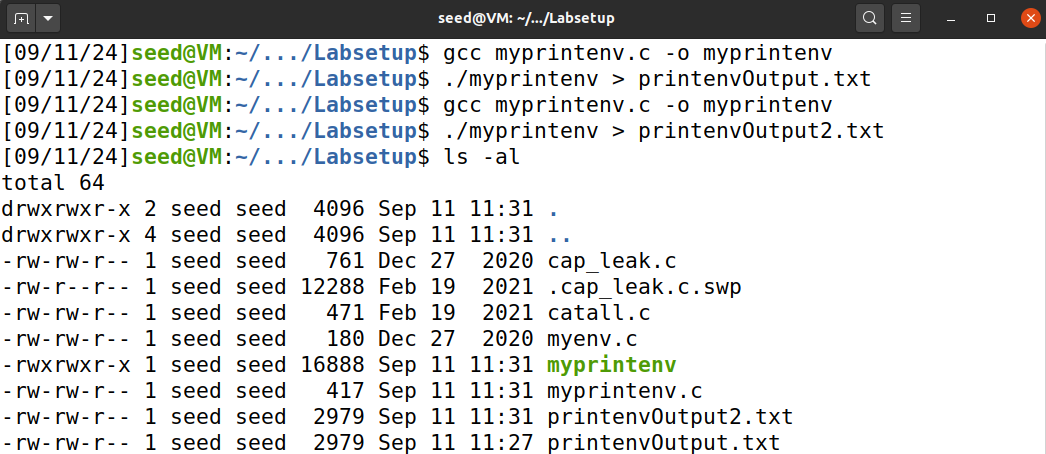
## Task 1:

 The above screenshot shows using the printenv command inside the terminal. The full output is not shown.

 The above screenshot shows using export and unset to export the shell variable name=Austin and then remove it.

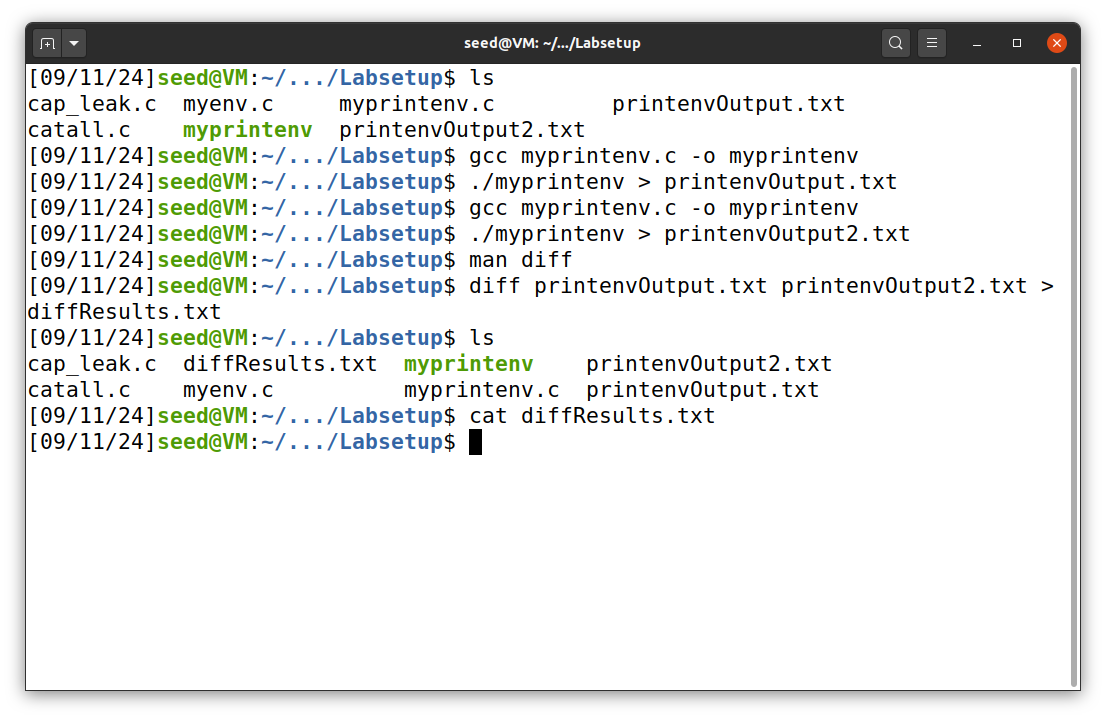
the command printenv name was used to prove its existence in the environment variables and its removal.

## Task 2:

 The above screenshot shows my process compiling and saving the output of the myprintenv program to printenvOutput.txt and printenvOutput2.txt for the modified myprintenv code which is shown below.

#include <unistd.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
extern char \*\*environ;  
  
void printenv()  
{  
 int i = 0;  
 while (environ[i] != NULL) {  
 printf("%s\n", environ[i]);  
 i++;  
 }  
}  
  
void main()  
{  
 pid\_t childPid;  
 switch(childPid = fork()) {  
 case 0: /\* child process \*/  
 //printenv();   
 exit(0);  
 default: /\* parent process \*/  
 printenv();   
 exit(0);  
 }  
}

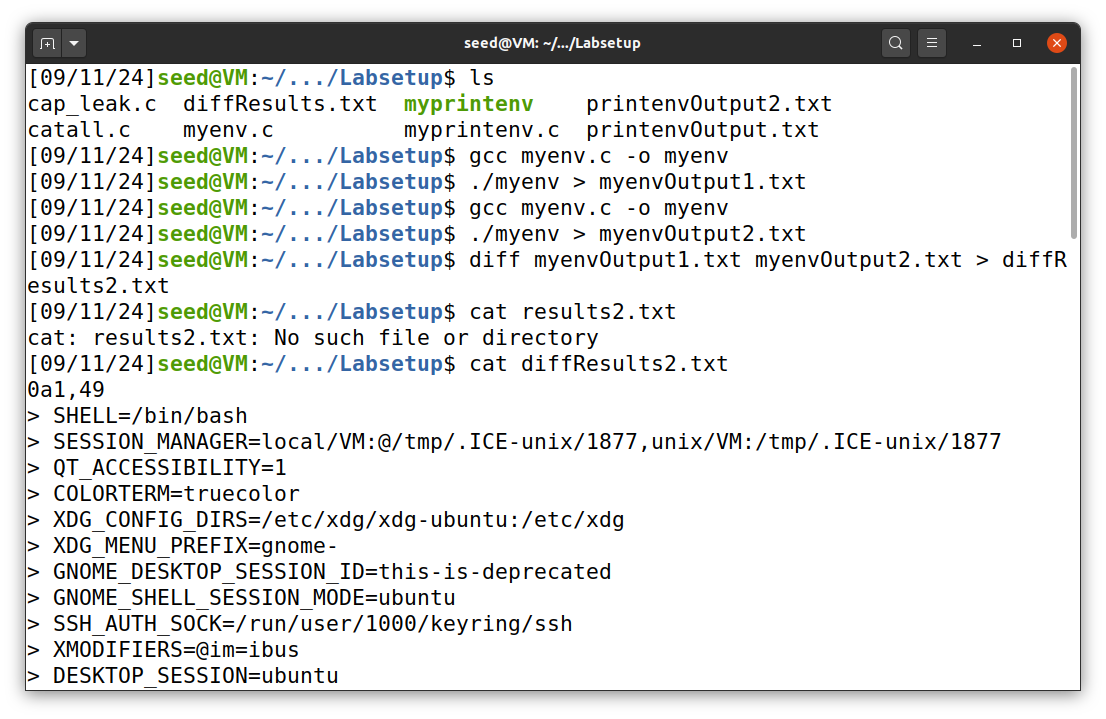
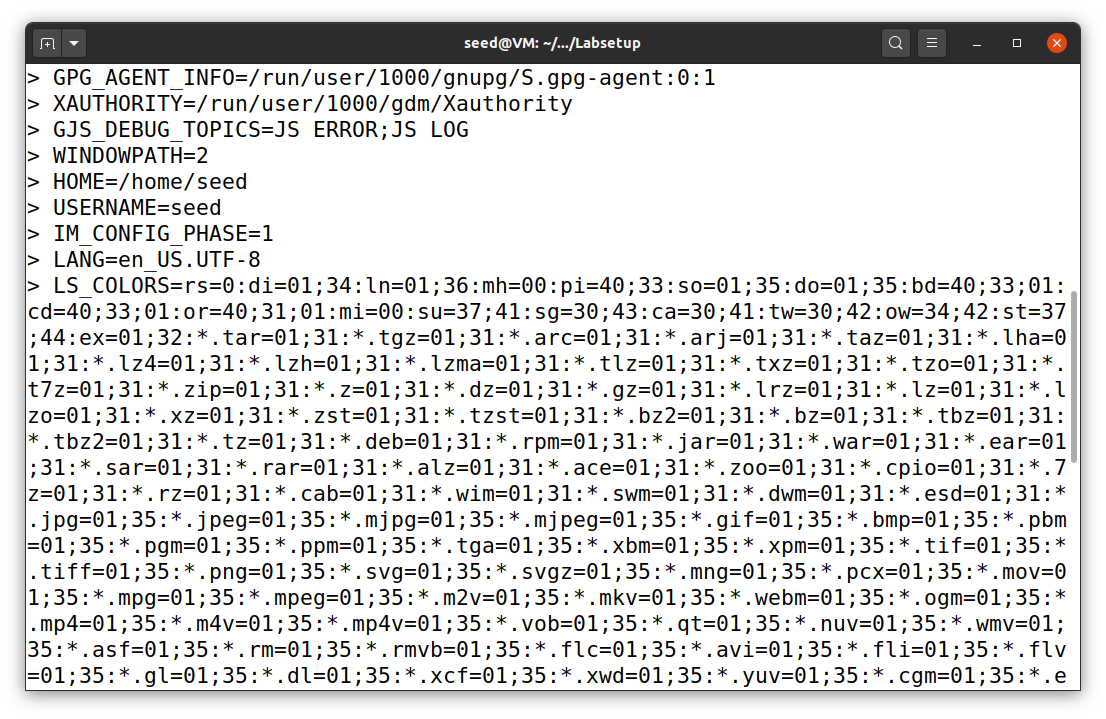
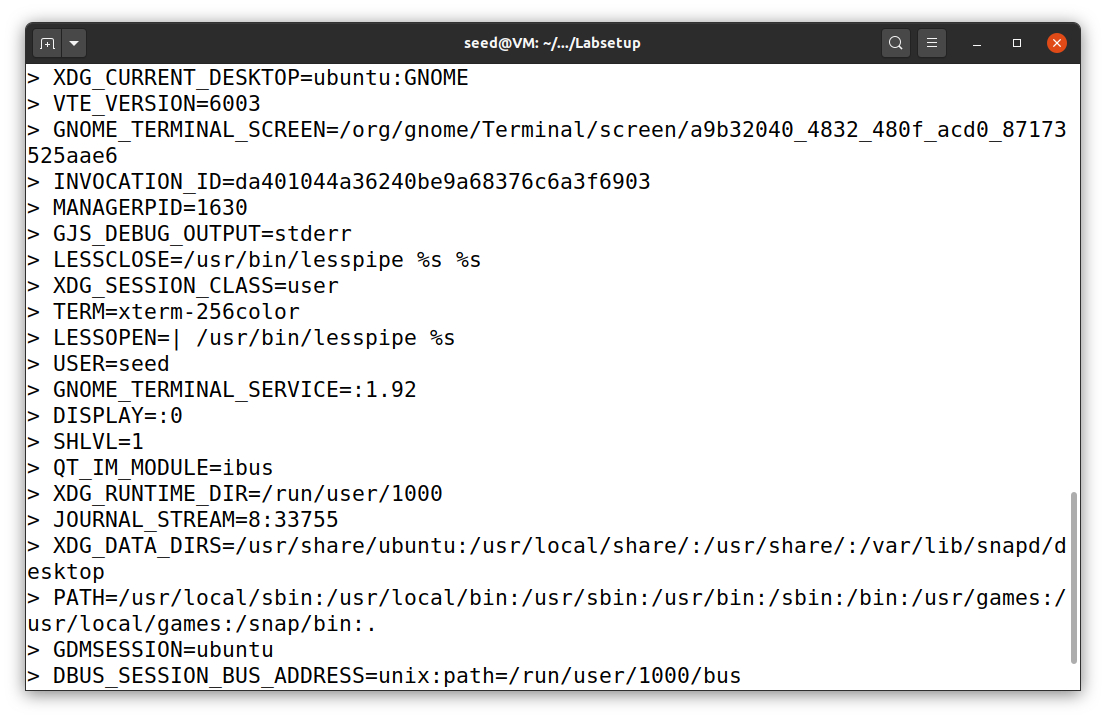
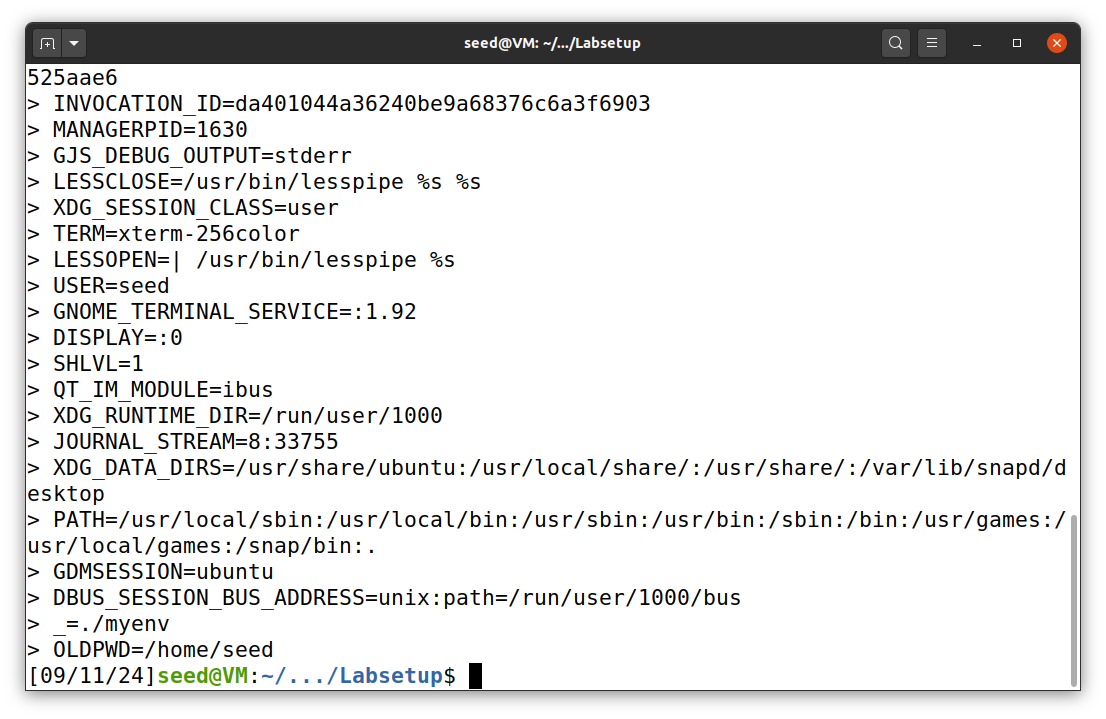
above is the modified code.

 The above screenshot shows the results of the diff. the output of the diff resulted in nothing which means that the printenv program prints out the same environment variables.

### Conclusions

Since there is no diff results, I conclude that in this current situation, the parent and child process have the same environment variables. The child process copies the parent process aswell as its environment variables.

## Task 3:

    The Above screenshots shows my process of compiling the code, running it, storing the results, and comparing the results with the changed code.

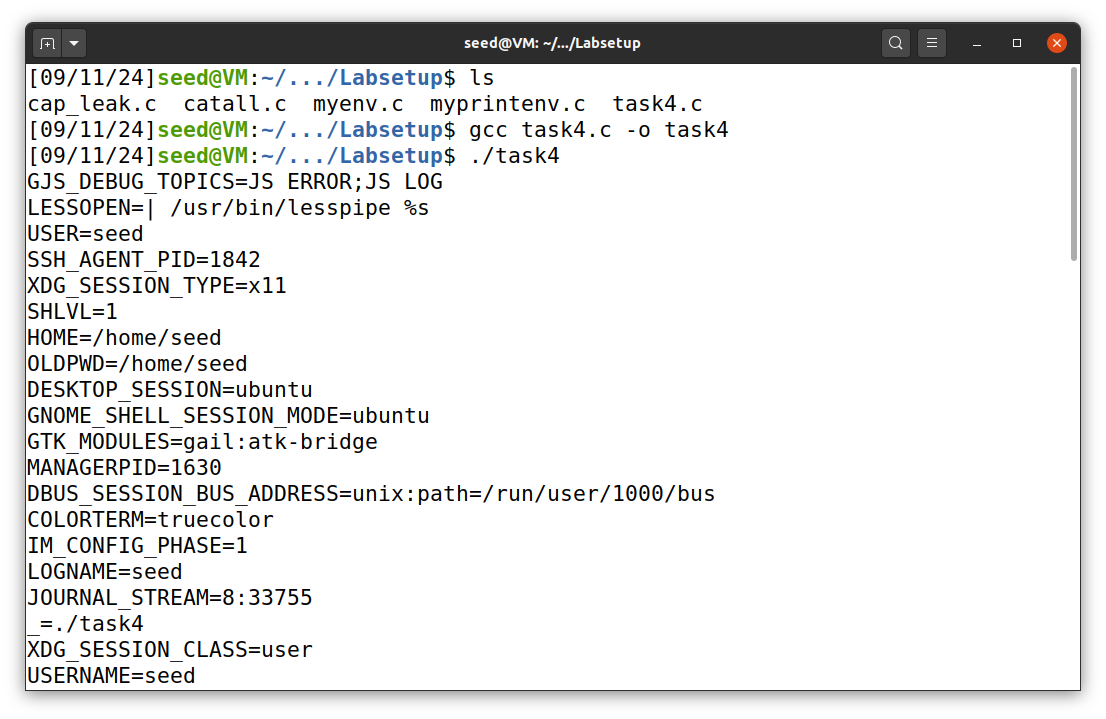
#include <unistd.h>  
  
extern char \*\*environ;  
  
int main()  
{  
 char \*argv[2];  
  
 argv[0] = "/usr/bin/env";  
 argv[1] = NULL;  
  
 execve("/usr/bin/env", argv, environ);   
  
 return 0 ;  
}

### Conclusions

The myenvOutput1.txt showed no info, but the myenvOutput2.txt showed the environment variables.

the environment variables are inherited by the new program if and only if they are explicitly passed in the execve() call.

## Task 4:

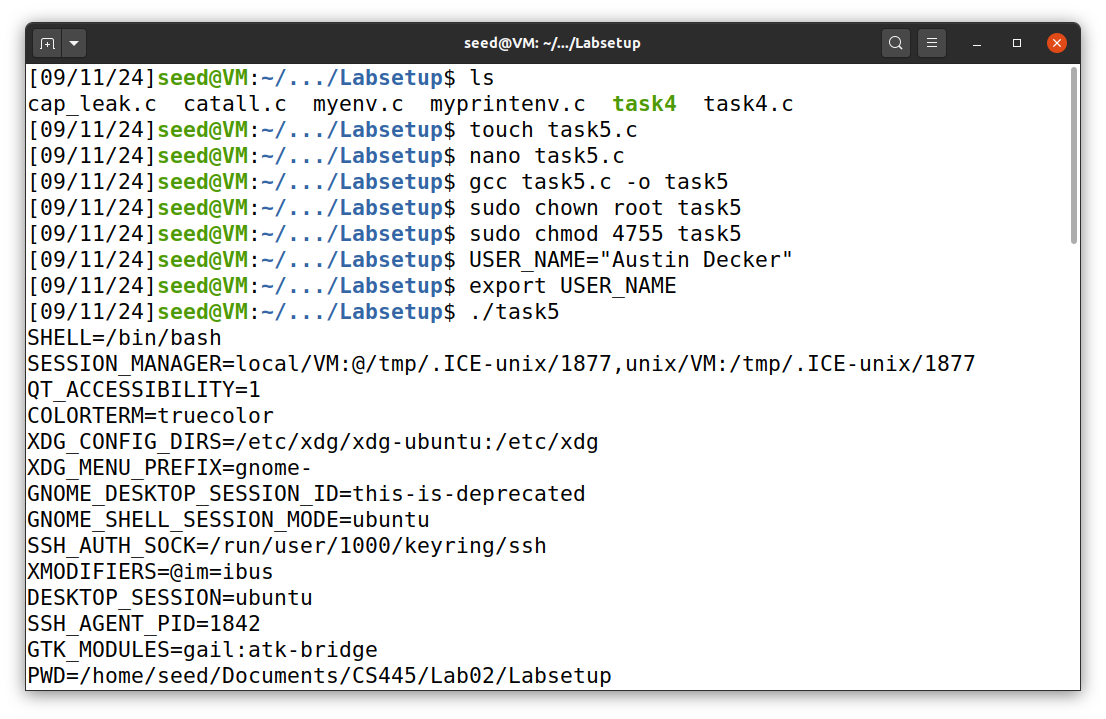
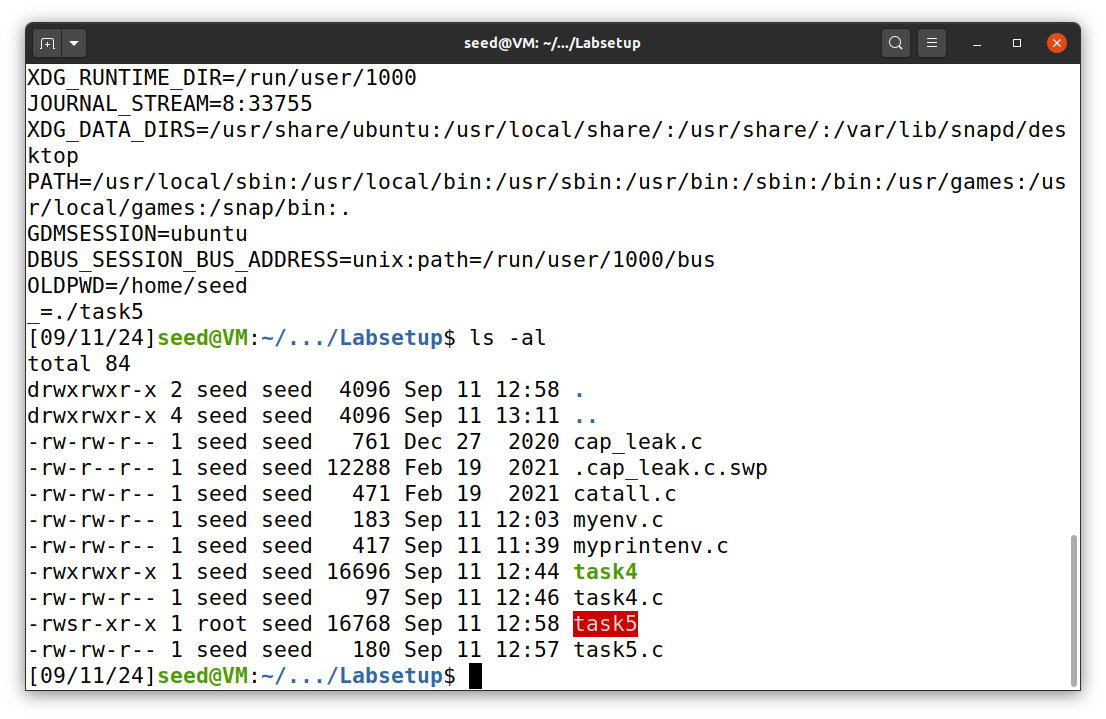


#include <stdio.h>  
#include <stdlib.h>  
int main()  
{  
 system("/usr/bin/env");  
 return 0 ;  
}

The above is my process for compiling and running the above code. The results verify that the process environment variables is passed to the process that is being runned by system() which is /bin/sh.

## Task 5:

#include <stdio.h>  
#include <stdlib.h>  
extern char \*\*environ;  
int main()  
{  
 int i = 0;  
 while (environ[i] != NULL) {  
 printf("%s\n", environ[i]);  
 i++;  
 }  
}

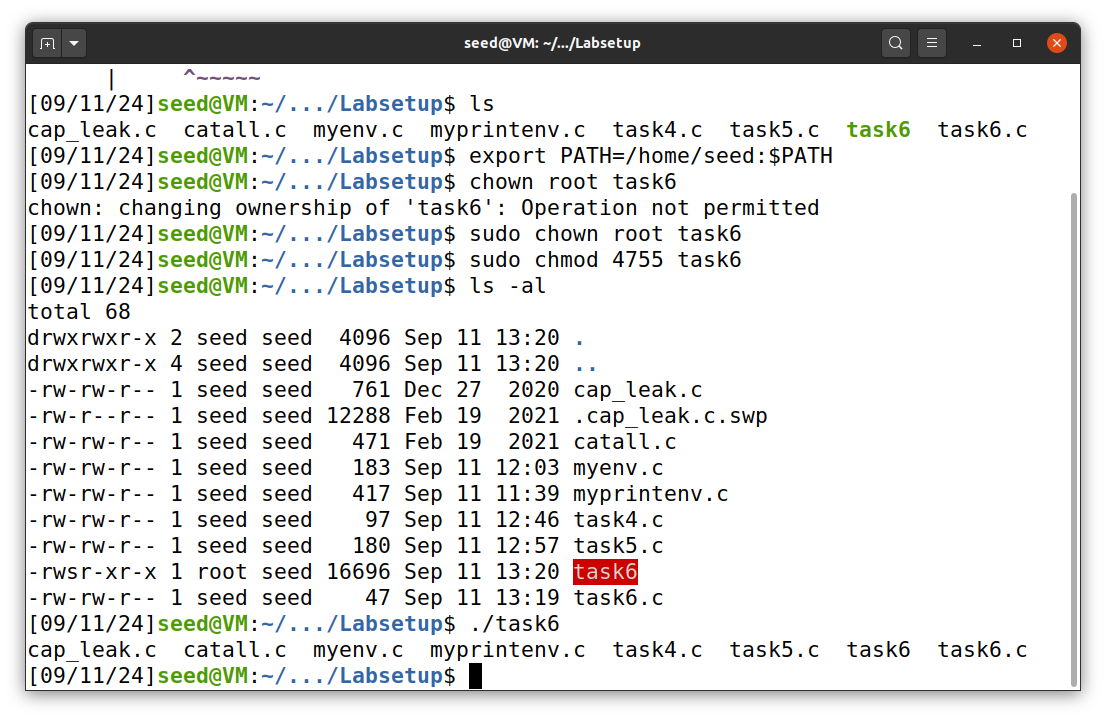
### Conclusions

The above screenshots show that the Environment Variable for user is the same despite that the program was run with setuid enabled. Notice my custom variable USER\_NAME="Austin Decker which was shown to be exported.

## Task 6:

Code used below:

int main()  
{  
 system("ls");  
 return 0;  
}

 The above screenshot shows my process for creating the executable, changing the owner to root, and setting the executable as a setuid program.

### Conclusions

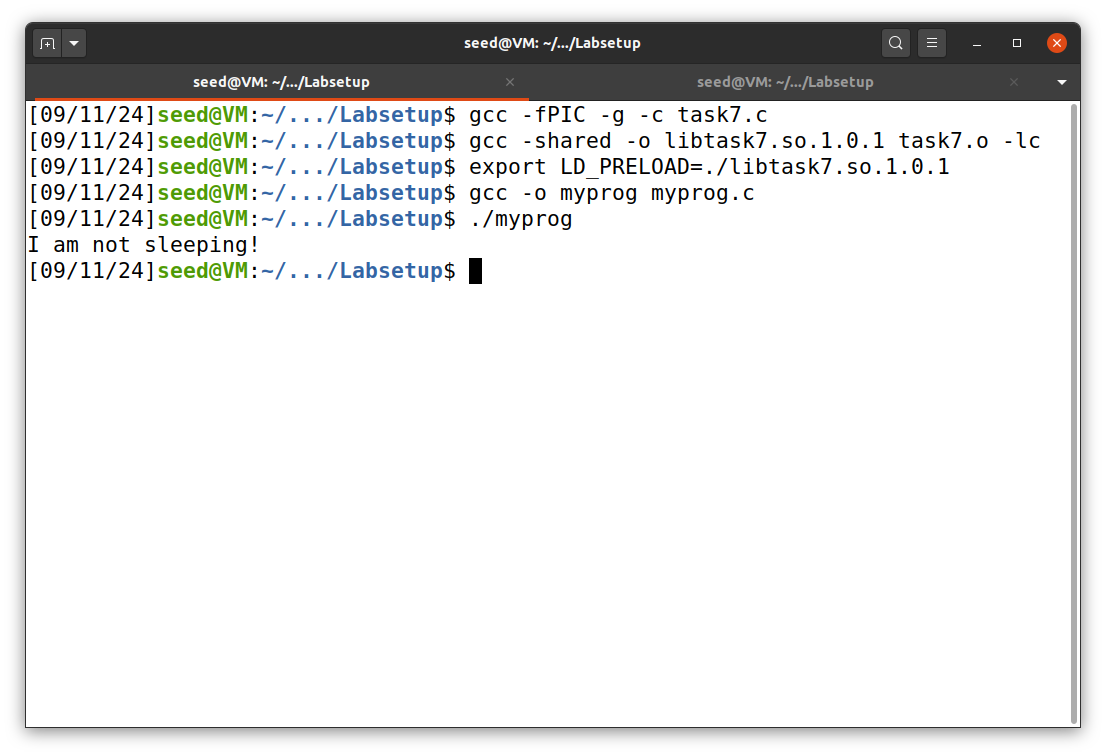
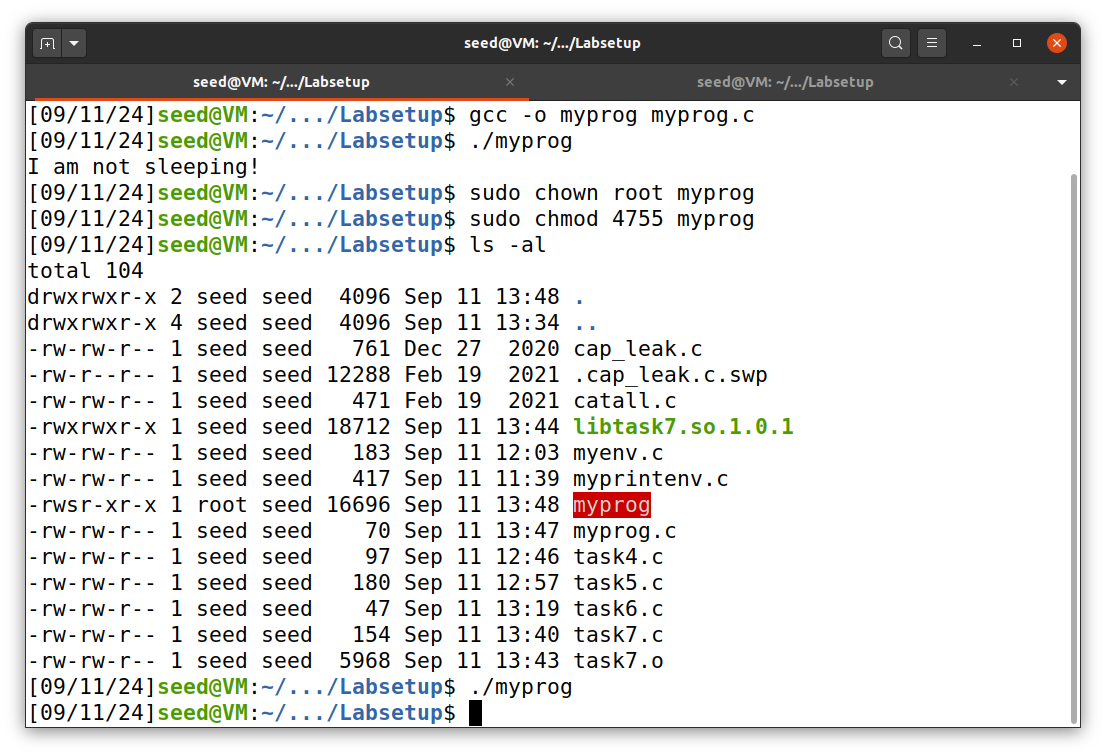
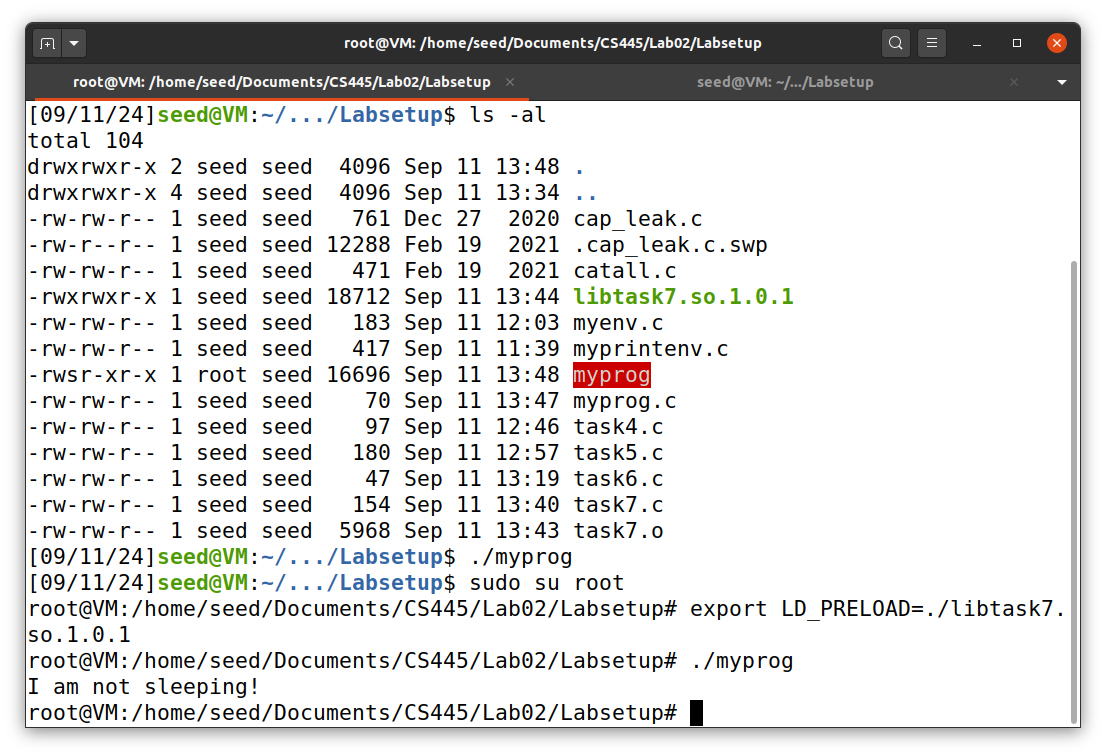
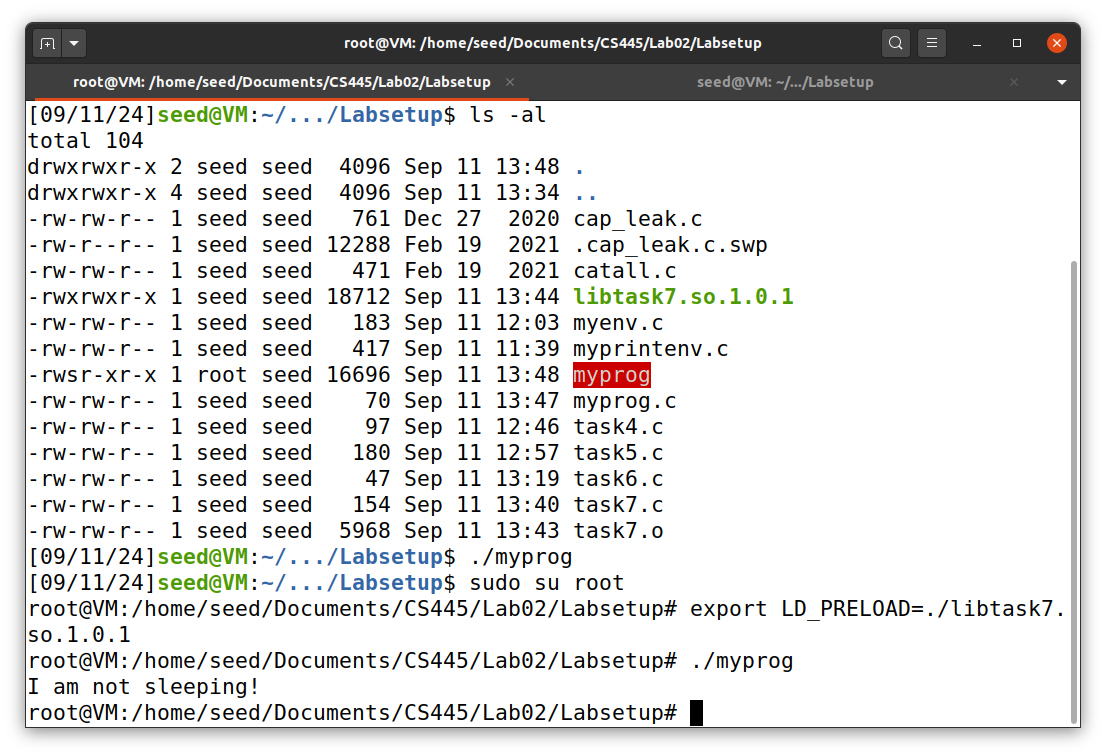
I cannot get the program to run my own code because there is a countermeasure in place within the called program /bin/sh from system().

## Task 7:

Code used below:

//task7.c  
#include <stdio.h>  
void sleep (int s)  
{  
/\* If this is invoked by a privileged program,  
you can do damages here! \*/  
printf("I am not sleeping!\n");  
}

/\* myprog.c \*/  
#include <unistd.h>  
int main()  
{  
 sleep(1);  
 return 0;  
}

* myprog run normally as a normal user 
* myprog run as setuid root program as a normal user 
* myprog run as setuid root with modified LD\_PRELOAD variable as root. 
* myprog run as setuid user1 as user1 with modified LD\_PRELOAD 

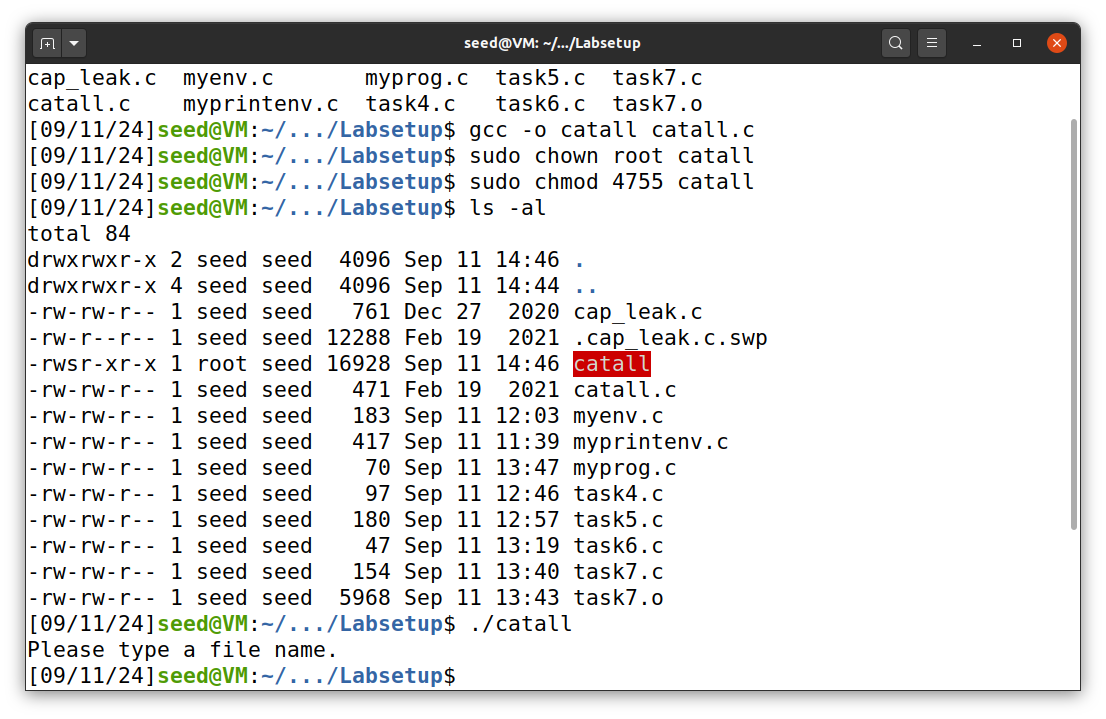
### Conclusions

The custom lib seems to only run when the Environment Variable LD\_PRELOAD is changed. if the setuid program owner does not have this custom lib linked with their environment variable, the bad code will not run. The owner must also have root privelege, otherwise the Environment Variable is ignored.

## Task 8:

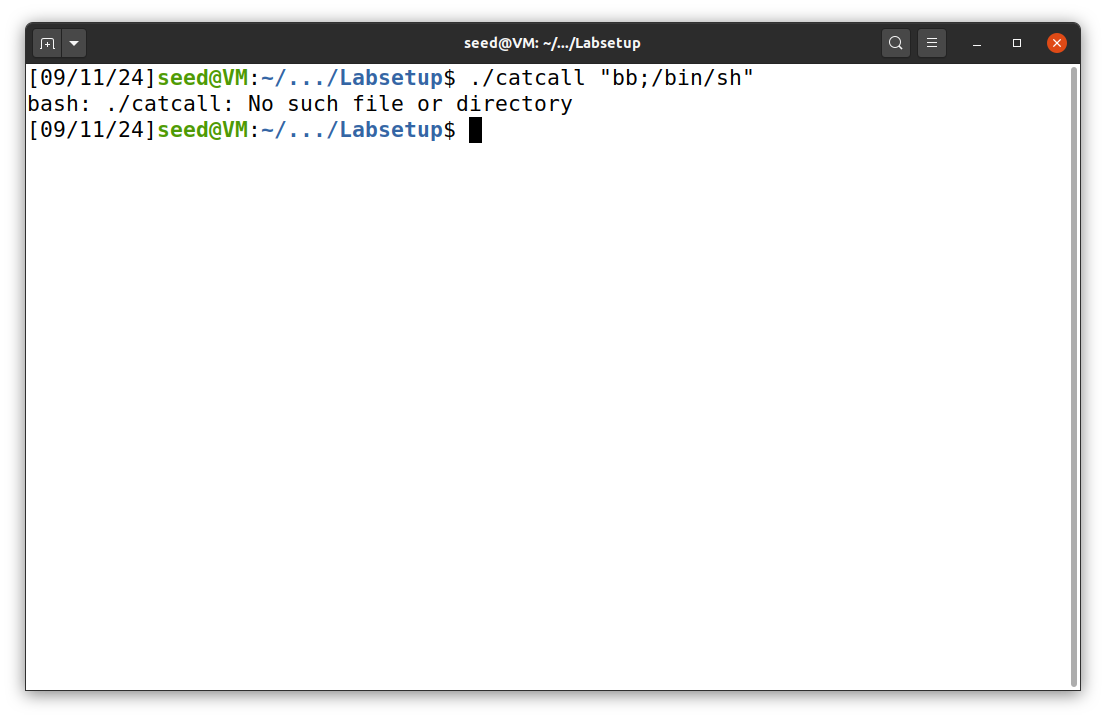
Code used:

#include <unistd.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(int argc, char \*argv[])  
{  
 char \*v[3];  
 char \*command;  
  
 if(argc < 2) {  
 printf("Please type a file name.\n");  
 return 1;  
 }  
  
 v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;  
  
 command = malloc(strlen(v[0]) + strlen(v[1]) + 2);  
 sprintf(command, "%s %s", v[0], v[1]);  
  
 // Use only one of the followings.  
 system(command);  
 // execve(v[0], v, NULL);  
  
 return 0 ;  
}

 above is the process for compiling and setting the program to setuid root.

### conclusion

one could get root shell access and modify anything, however there is a countermeasure in ubuntu that currenty prevents it without switching /bin/sh to /bin/zsh.

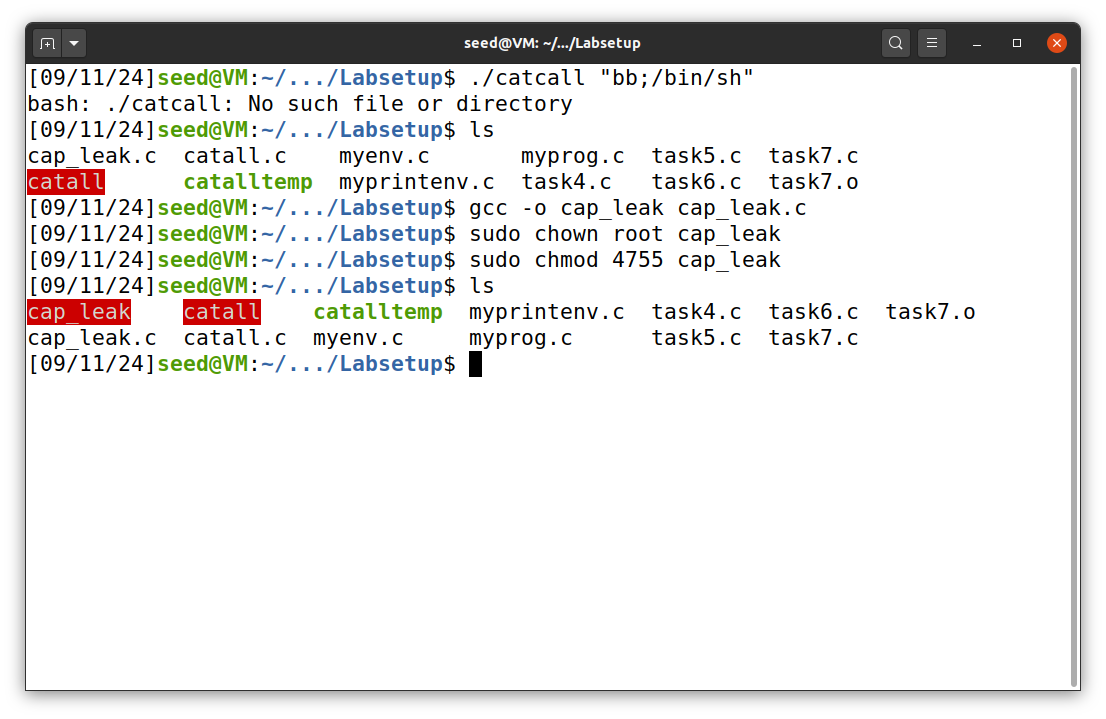


Using execve() would prevent the current program from being abused with the vulnerability in system() if the countermeasure was not in place.

## Task 9:

Code Used:

#include <unistd.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <fcntl.h>  
  
void main()  
{  
 int fd;  
 char \*v[2];  
  
 /\* Assume that /etc/zzz is an important system file,  
 \* and it is owned by root with permission 0644.  
 \* Before running this program, you should create  
 \* the file /etc/zzz first. \*/  
 fd = open("/etc/zzz", O\_RDWR | O\_APPEND);   
 if (fd == -1) {  
 printf("Cannot open /etc/zzz\n");  
 exit(0);  
 }  
  
 // Print out the file descriptor value  
 printf("fd is %d\n", fd);  
  
 // Permanently disable the privilege by making the  
 // effective uid the same as the real uid  
 setuid(getuid());   
  
 // Execute /bin/sh  
 v[0] = "/bin/sh"; v[1] = 0;  
 execve(v[0], v, 0);   
}

Setting up executable 

Capabiliy leak vulnerability 