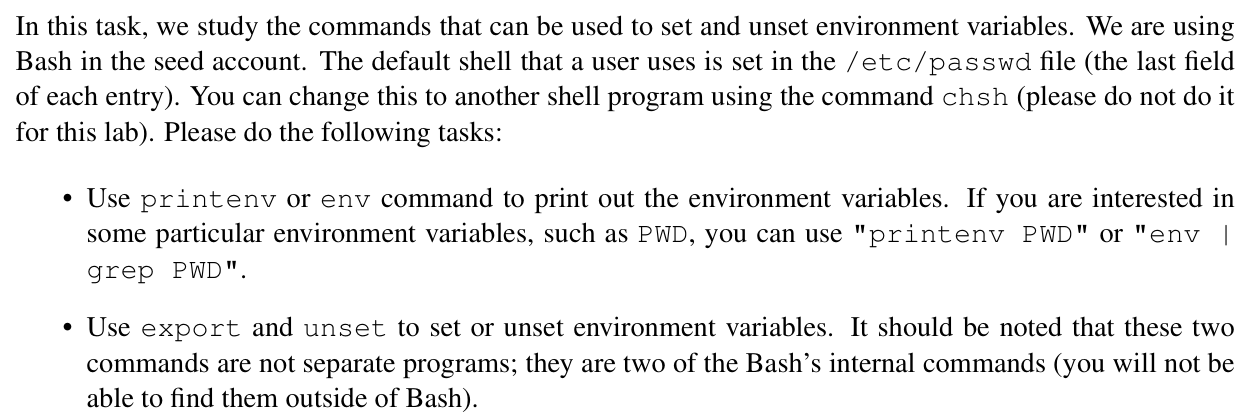


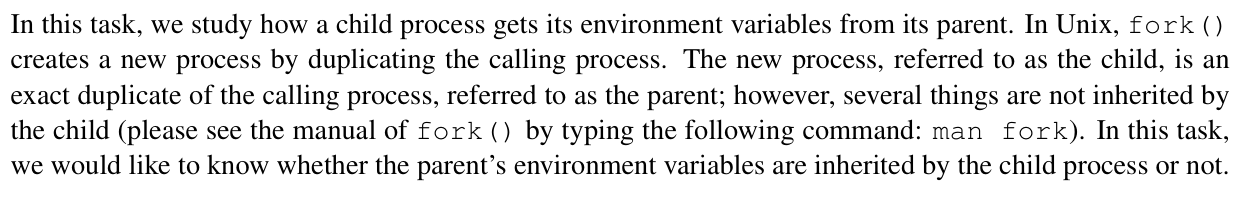
**Task 1: Manipulating Environment Variables**



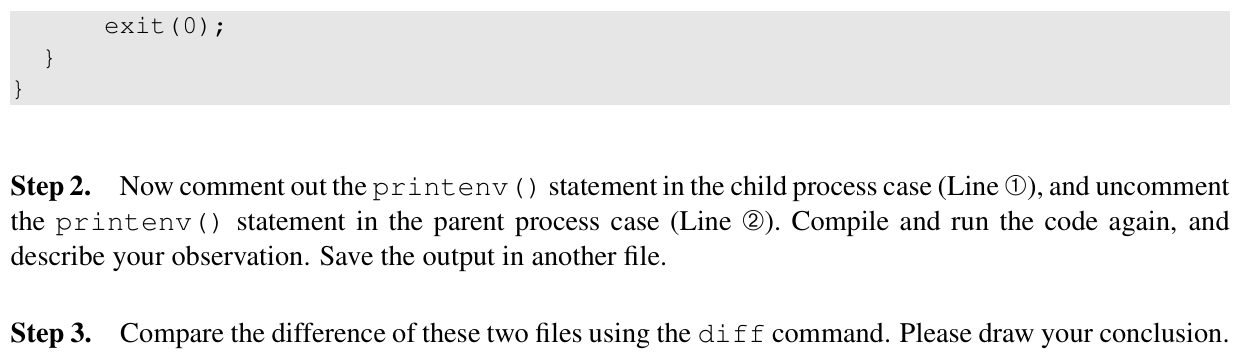
|  |
| --- |
| Use “env” command to print out the environment variables |
| Use “printenv PWD” command, which only show the value of PWD variable |

• Use export and unset to set or unset environment variables. It should be noted that these two commands are not separate programs; they are two of the Bash’s internal commands (you will not be able to find them outside of Bash).

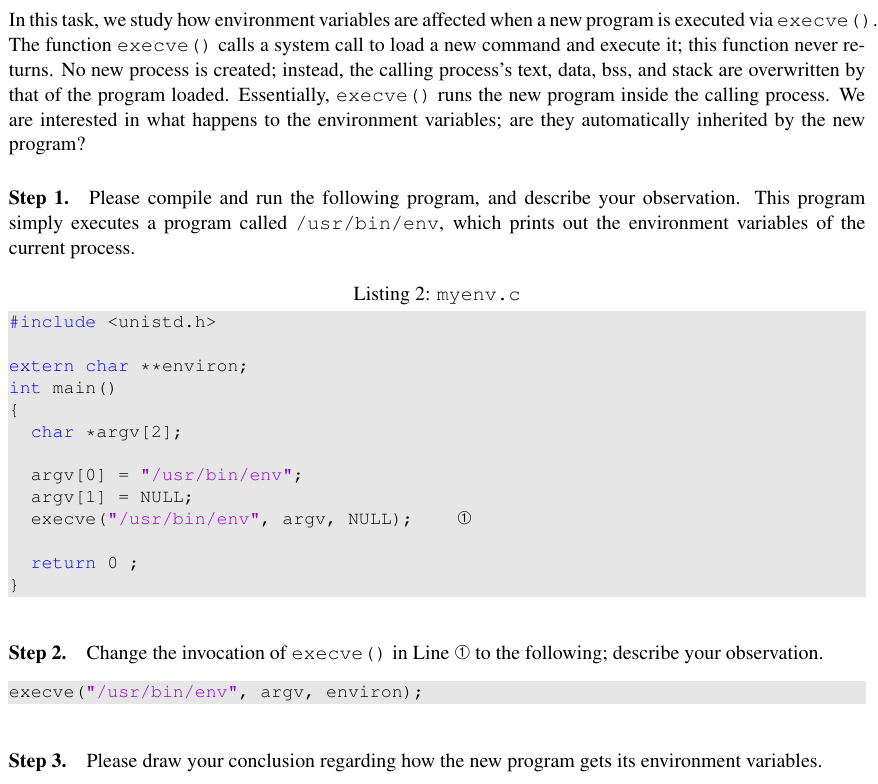
|  |
| --- |
| Use “export” command to create an environment variable named “myenvvar”, with the value “My new env variable”. Then use “printenv” command to check the value of “myenvvar” |
| Use “unset” command to remove the “myenvvar” variable, and then use “printenv” command to check if ”myenvvaar” still exists. (Bash shows nothing after the unset command.) |

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

****

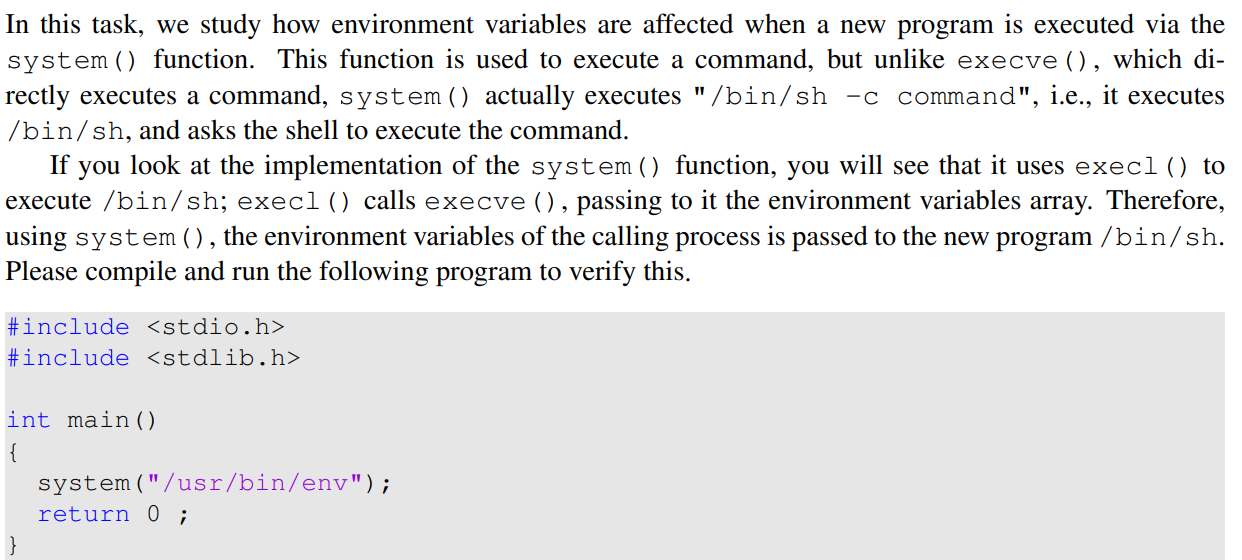
****

|  |
| --- |
| Step1:    Use gcc compile “myprintenv.c” and generate a file called “a.out” in the Labsetup folder.    Use “a.out > file” to save the output to “file” file. The right side is the content of “file” |
| Step2    Use gcc compile edited “myprintenv.c”. Run “a.out” and save the output to “another” file. |
| Step3    Use “diff” command to compare the “file” and “another”.  The terminal show no different between the two files.  Since the child process inherit a copy of parent environment variables. |

**Task 3: Environment Variables and execve() In this task, we study** 

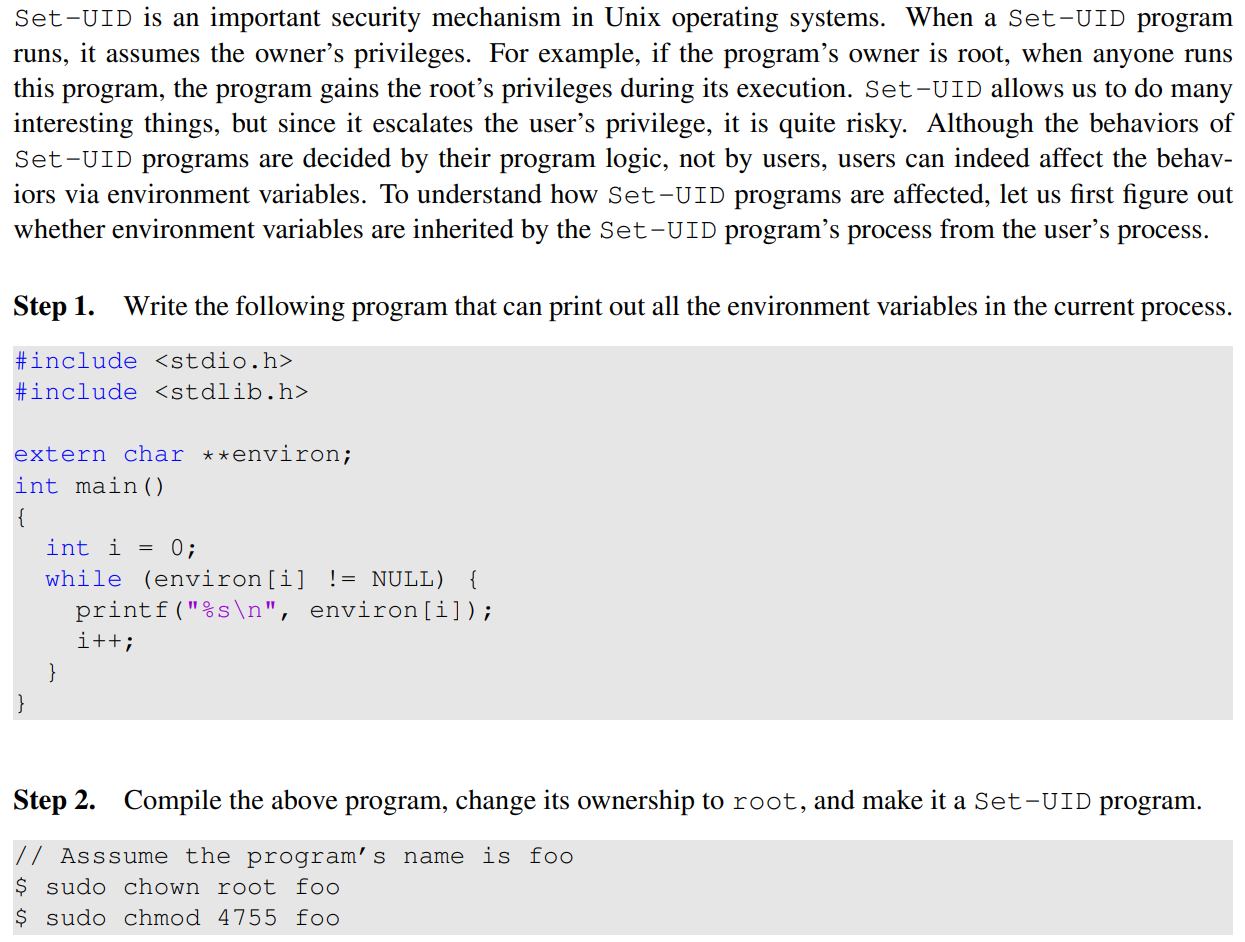
|  |
| --- |
| **Step1 & Step2**    Use gcc compile myenv.c and name output file as “myenv”  Use gcc compile edited myenv.c and name output file as “myenv2”  myenv no output, and myenv2 show the environment variables. |
| **Step3**  The unedited mynev.c (line12: execve("/usr/bin/env", argv, NULL); )  NULL is used as the third argument of “execve”, resulting in no environment variables being passed to the new process.  After edited (line12: execve("/usr/bin/env", argv, eviron);) “eviron” is used as the third argument of execve, so can the new process get the environment variable which the current process keep. |

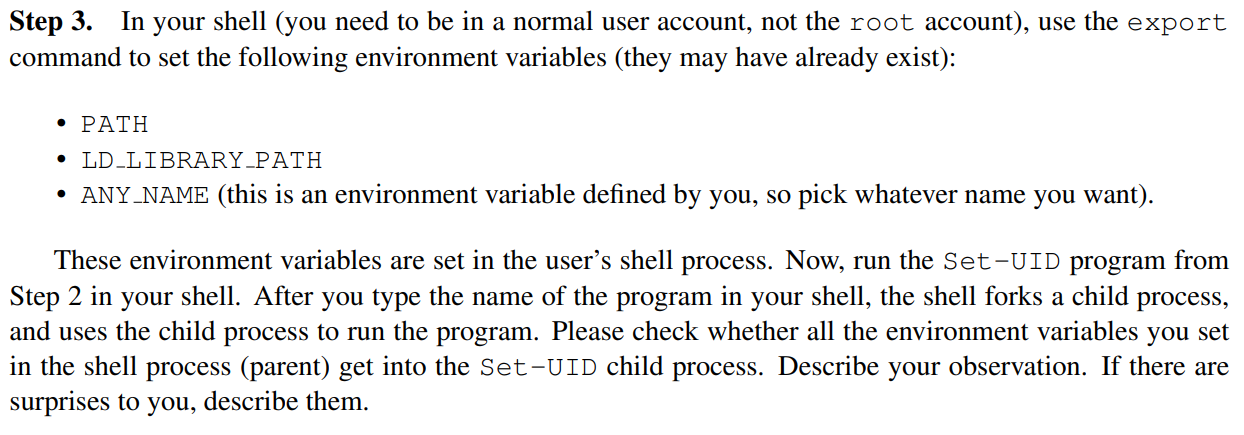
**Task 4: Environment Variables and system()**



|  |
| --- |
| Use touch command to create task4.c and write the given code in it.  Use gcc compile task4.c and run.  The output show the environment variables of the current process.  Since the system function pass the environment variables to /bin/sh implicitly |

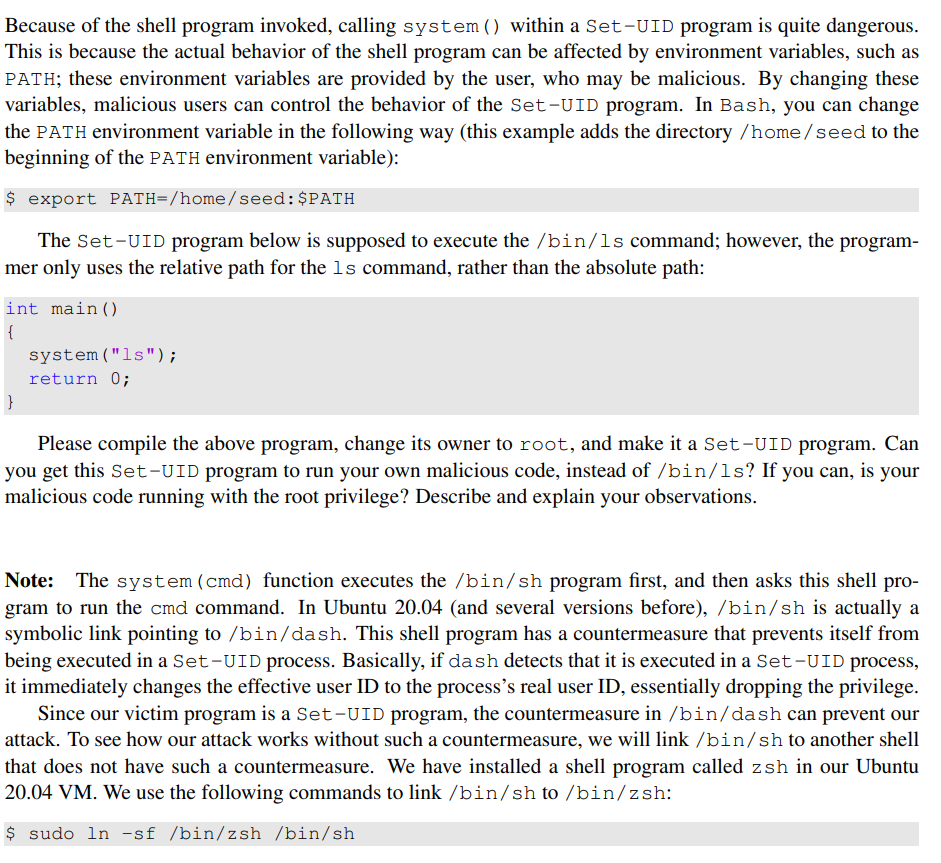
**Task 5: Environment Variable and Set-UID Programs**

****

****

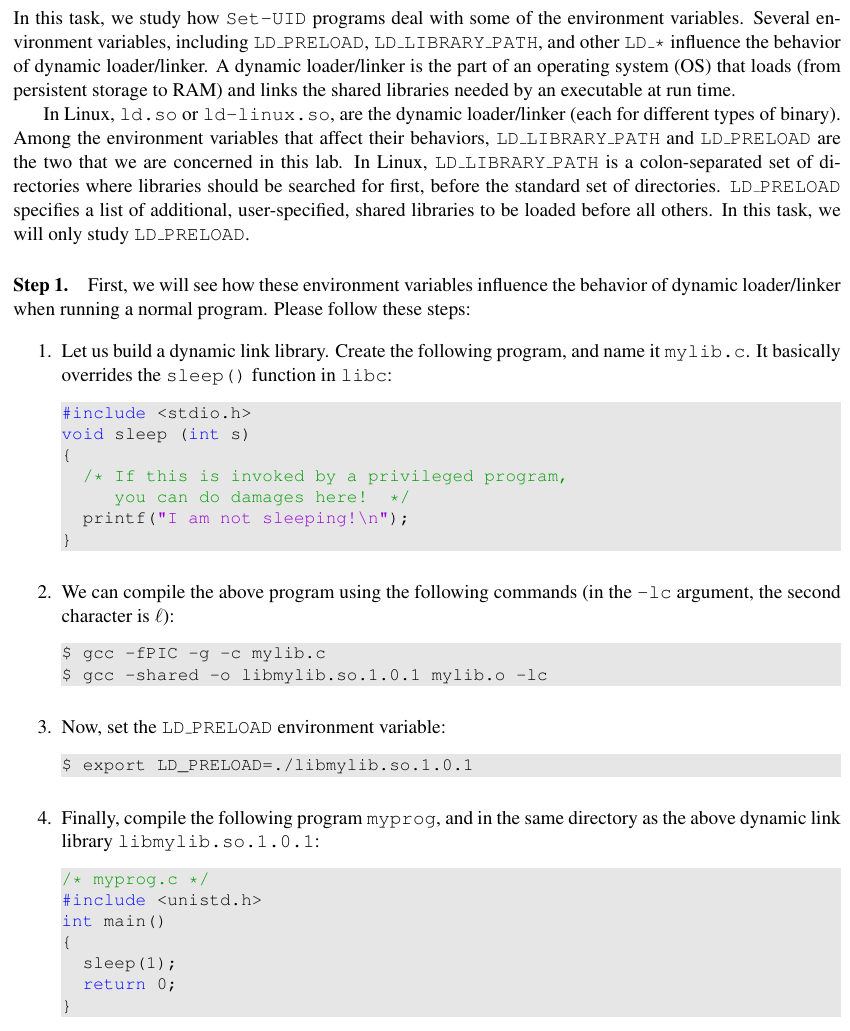
|  |
| --- |
| **Step1**    Use touch command to create task4.c and write the given code in it.  Use gcc compile task4.c and run.  The output show the environment variables of the current process. |
| **Step 2**    "sudo chown root task5" making the owner of task5 is root  "sudo chmod 4755 task5" making the program a SET-UID program |
| **Step3**    Use whoami to check current user  Use “export” command to create an environment variable named “myenvvar”.    **task5 | grep myenvar, check “myenvvar” if exist -> Yes**  **task5 | grep PATH, check “PATH” if exist -> Yes**  **task5 | grep LD\_LIBRARY\_PATH, check “LD\_LIBRARY\_PATH” if exist -> No**  **LD\_LIBRARY\_PATH is missing, since real user id and effective user id are different.** |

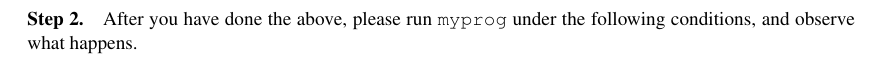
**Task 6: The PATH Environment Variable and Set-UID Programs**

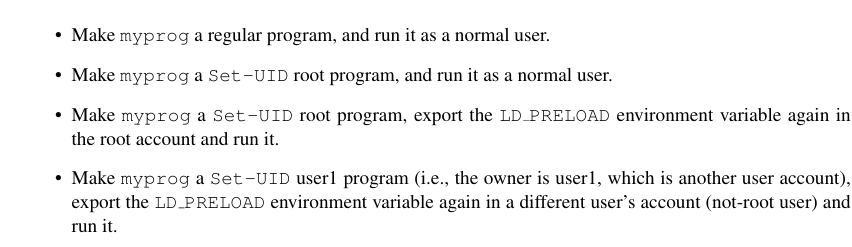
****

|  |
| --- |
| Create a file “task6.c” (compile the code above), compiled by gcc)    Create a program used as “malicious code”    (I have moved all the needed file to “/home/seed” )    Use sudo ln -sf /bin/zsh /bin/sh link sh to zsh  Change its owner to root, make it a Set-UID program and check whether it is a Set-UID program.  Try to run my code instead of “ls” program, I changed the value of environment. Variable PATH. (making the program to search for the file in my folder first)  After run task 6, the output is “My malicious code is launched !”  It run my code instead of the “/bin/ls” program !!  Conclusion:  Changing the PATH to a specific folder may execute hacker’s code.  Using system(“…”) with relative path in a Set-UID program is dangerous, allowing attacker perform his code. |

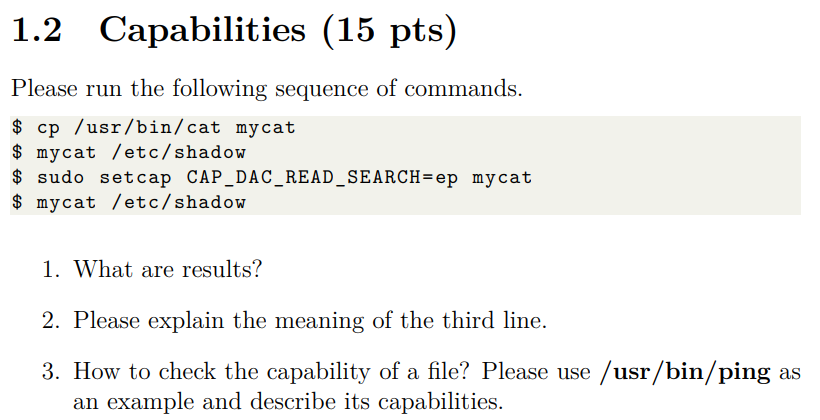
**Task 7 The LD\_PRELOAD Environment Variable and Set-UID program**

****

****

****

|  |
| --- |
| **Step 1**    Create a file mylib.c and compiled by gcc  -fIPC means position independent code  -g means output debugging information  -c means compile the file but not link it  -shared means a shared object which can be linked to other object |
| **Step2**    A regular program “myprog”, run it as a normal user  Output “I am not sleeping”    Make it a Set-UID program, and run it as a normal user  The terminal stuck 1 second then output nothing.    Make it a Set-UID program, and run it as root |

****

|  |
| --- |
| 1. **What are results** |
| 1. **Please explain the meaning of the third line.**   Set capability on the “mycat” with superuser permission.  Set ‘CAP\_DAC\_READ\_SEARCH’ to ep (effective & permit)  CAP\_DAC\_READ\_SEARCH : Bypass file read-only operation checks  “mycat” now can read the file although he/she is not root |
| 1. **How to check the capability of a file?**   **Use ‘getcap’ command**  **Ex:** |