**LAB ASSIGNMENT 5**

**Assignment - Environment Variables**

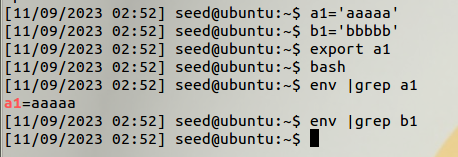
1. What is the difference between environment variables and shell variables?

Ans:

Sure, here is the information in a table format:

|  |  |  |
| --- | --- | --- |
|  | **Shell Variables** | **Environment Variables** |
| **Definition** | Shell variables are specific to the shell session in which they are defined | Environment variables are a type of shell variable that has been exported |
| **Scope** | They are used to keep track of data in the current shell session | They are visible not only in the shell session that created them, but also for any process (not just shells) that are started from that session |
| **Inheritance** | They are not inherited by child processes | They are available system-wide and can be accessed by any process or program running on the system |
| **Naming Convention** | Shell variables usually have names with lower-case letters | They often have names consisting of upper-case letters |
| **Example** | If you’re running another application from the shell, that application will not inherit the shell variable | A typical case of this is the PATH environment variable, which may be set in the shell and later used by any program that wants to start programs without specifying a full path to them |

A screenshot of my terminal:

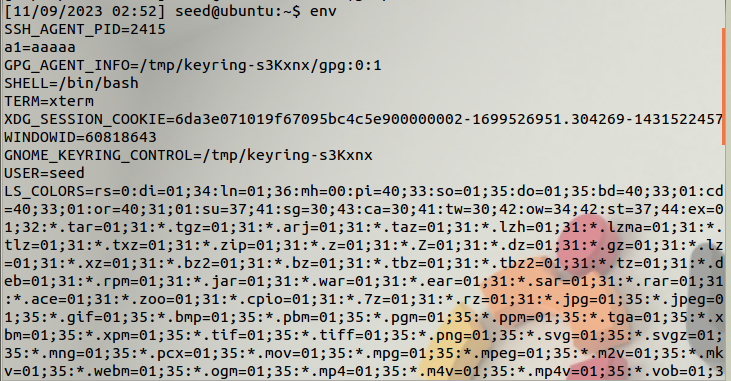


In the above screenshot we can see that a1 is a global variable while b1 is a local variable. Inside the child process only **a1** is accessible as an env variable, **b1** shows no output.

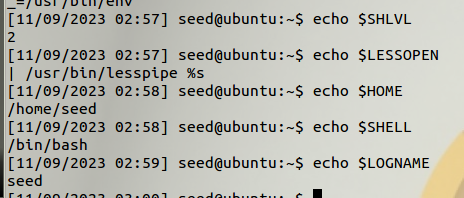
1. Display the environment variables currently available to the shell. Display the contents of any 5 such variables. (env command)

All the env variables in the system can be seen using the env cmd. We can check the content of it either using **echo $varname** or **env |grep varname**, here we have used the echo cmd.

Looking at all the environment variables: command: **env**



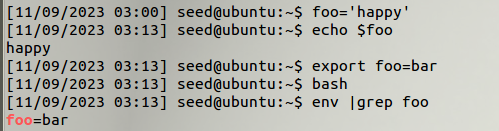
Printing any 5 environment variables:



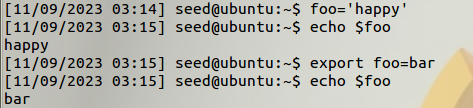
1. In Bash, if we run "export foo=bar", does it change the environment variable of the current process?

Ans: Yes the value changes inn both the current process and child process.

For child process:



For current process:



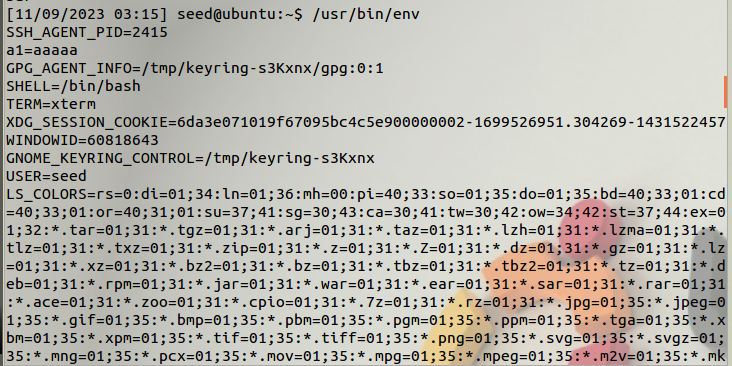
Here we see that the initial value of foo was **happy** but once we do export it’s value changes to **bar.**

1. The followings are two different ways to print out environment variables. Describe their differences:

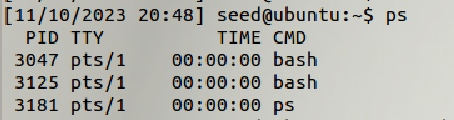
• $ /usr/bin/env

• $ /usr/bin/strings /proc/$$/environ

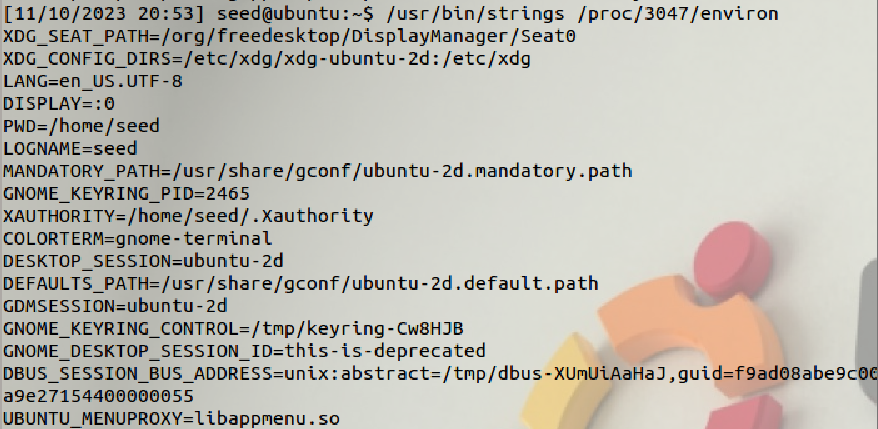
Ans: **$ /usr/bin/env** – this displays all the current environment variables. It prints the environment variables in the format **VARNAME=value** on separate lines. This command is versatile and can also be used to run a command with a modified environment. For example, **env VARNAME=value command** will run 'command' with an environment variable set to a specific value.



* **extracting some pids**



**$ /usr/bin/strings /proc/$$/environ**- This method directly accesses the **/proc** filesystem, specifically the **environ** file for the current process (via the **$$** variable, which represents the PID of the current shell). The **strings** command extracts readable text from binary data, and in this case, it extracts the environment variables. This method directly reads the process environment, so it might include variables that are not part of the typical shell environment.



1. Write a program to pass user defined environment variables using the help of command line arguments and display the same.

**The program:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Declare extern char \*\*environ to access environment variables

extern char \*\*environ;

int main(int argc, char \*argv[]) {

// Check if there are user-defined environment variables

if (argc < 2) {

printf("Usage: %s VAR1=value1 VAR2=value2 ...\n", argv[0]);

return 1;

}

// Iterate through the provided variables

for (int i = 1; i < argc; ++i) {

// Split each variable into name and value

char \*variable = argv[i];

char \*equals\_pos = strchr(variable, '=');

if (equals\_pos != NULL) {

// Extract name and value

\*equals\_pos = '\0'; // Null-terminate at '=' to separate name and value

char \*name = variable;

char \*value = equals\_pos + 1;

// Set the environment variable

if (setenv(name, value, 1) != 0) {

fprintf(stderr, "Failed to set environment variable: %s\n", name);

return 1;

}

} else {

fprintf(stderr, "Invalid format for environment variable: %s\n", variable);

return 1;

}

}

// Display the environment variables

printf("User-defined environment variables:\n");

char \*\*env;

for (env = environ; \*env != NULL; ++env) {

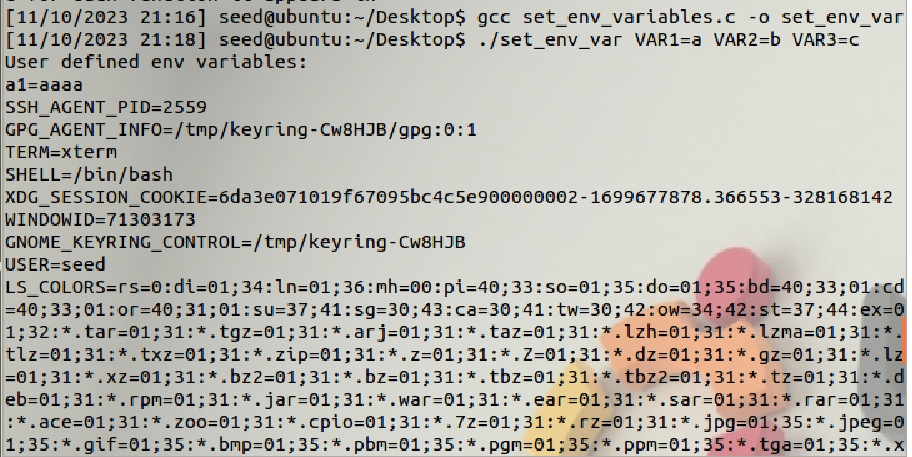
printf("%s\n", \*env);

}

return 0;

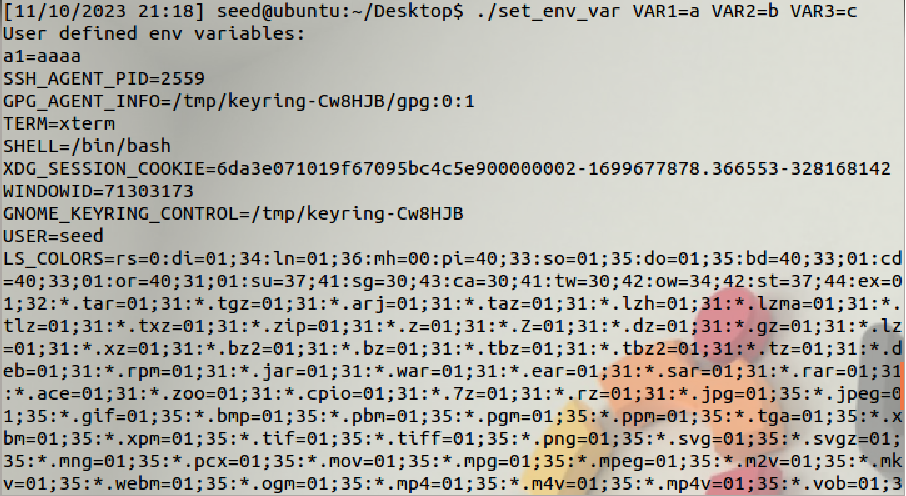
}

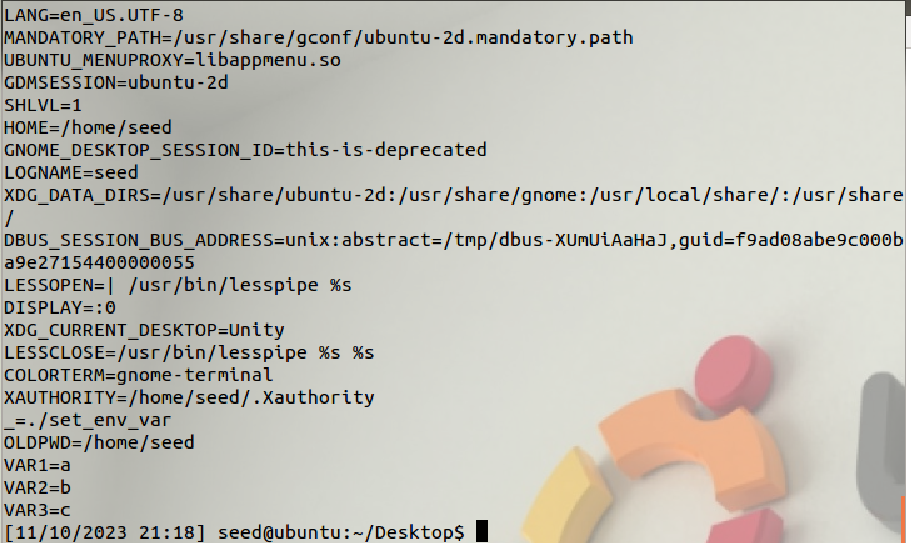
Running the code:



**OUTPUT:**

It prints all the current env variables along with the new env variables.

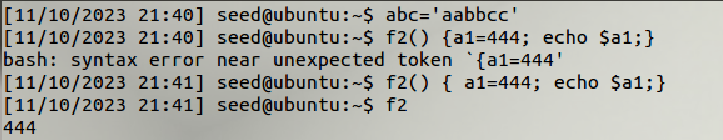




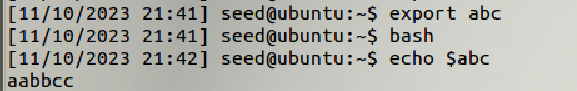
As we can see VAR1,VAR2,VAR3 are the new variables declared as environment variables on running the program.

1. Experiment passing both variables and functions as environment variables to the child process.

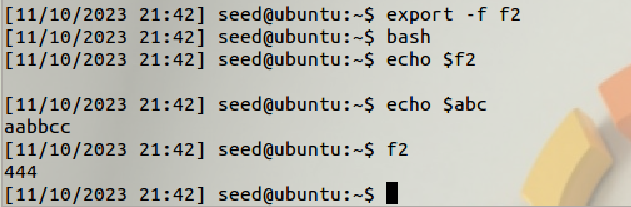
***$ Creating variables and functions***



***$Passing variable as env variable***



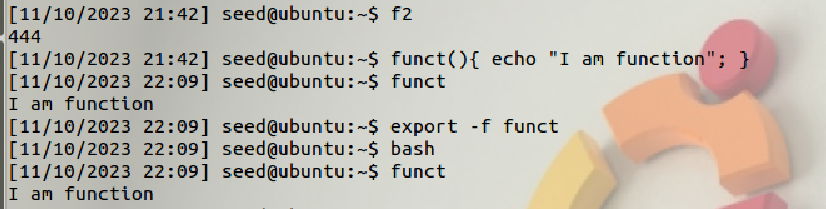
***$Passing function as an environment function***



As we notice both the function and the work run as environment variable in the child process initiated by **“bash”** command.

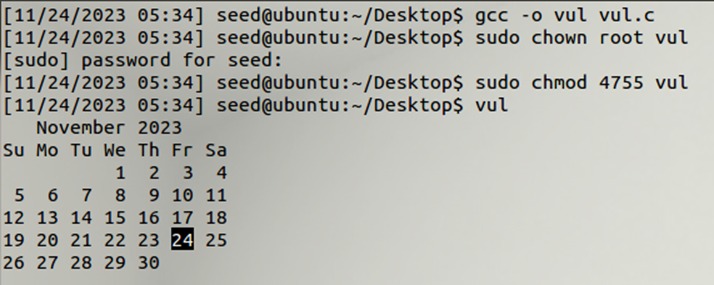
1. Define a function inside an environment variable. When executing a child process how can we invoke that function by passing that as an environment variable to child process.

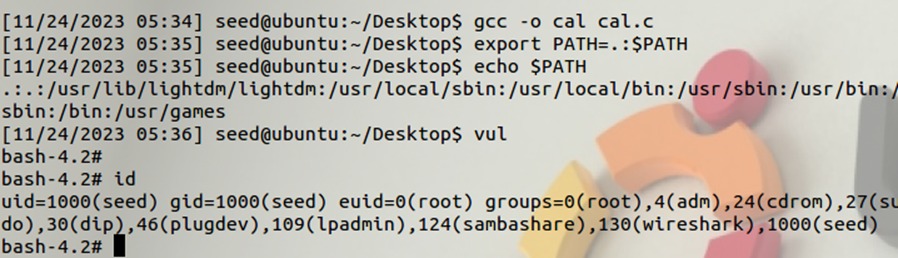
Ans: Same as above. Let’s show another example



1. Demonstrate the redirection of an inbuilt command or an application (e.g. ls, cat, cal or any command of your choice) to some user defined executable by manipulating the required environment variable.

ANS:

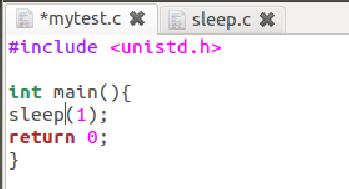




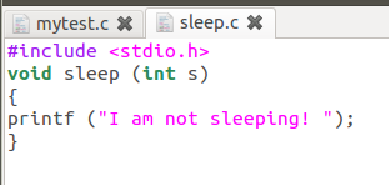
1. Write a program to demonstrate the attacks through the use of Dynamic Linker. Also, show the effect if we set the same program as a SetUID program

ANS:

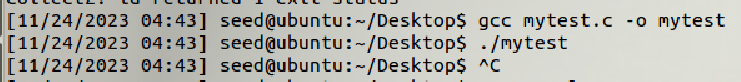
Creating a file that is using a SETUID program:



Creating a file to escape the setUID and run something else:



Running mytest.c without changing the sleep(1) function:



As we can see the sleep function is working and to escape the sleep I use ctrl+C.

Next we make sleep.c that we created act like the sleep function.

To do this we create sleep.o then using LD\_PRELOAD variable we make sleep use the sleep.c file that we just created instead of the actual SetUID sleep.

