Lab 2.1. Introduction to UNIX Systems Programming

Objectives

In this lab we will study the basic use and conventions of a UNIX system API and its development environment. In particular, we will use functions for error management and getting system, user and system time information.

Contents

Lab Environment

This lab only requires the development environment (compiler, editor and debugger). These tools are available in the ASOR VMs as well as in the physical machine of the lab.

Any editor can be used (`vi`, `nano`, `xedit`…). Also, it is also possible to use C or C++ language (`gcc` or `g++` compiler, respectively). If several files have to be compiled, the use of `make` is recommended. Finally, the recommended debugger is `gdb`. The use of IDEs like Eclipse **is not allowed**.

Error Management

Use available functions in the system API to manage errors in the following cases. For each exercise, add the needed header files (`#include`). The use of `perror(3)` and `strerror(3)` calls is recommended.

Exercise 1. Add the needed code to appropriately manage the error generated by the `setuid(2)` call. Using the `man` page, consult the purpose of this call and its prototype.

```
int main() {
    setuid(0);
    return 1;
}
```

Exercise 2. In the previous code, print the error code generated by the call, both the number and the associated a string.

Exercise 3. Write a program that prints all error messages available in the system. Initially, consider that the limit of possible errors is 255.

System Information

Exercise 1. System command `uname(1)` shows information about several aspects of the system. Check the `man` page and get information about the system.

Exercise 2. Write a program that shows every aspect of the system and its value in a clear way. Check the correct execution of the call on every case. Check the `man` page of `uname(2)`.

Exercise 3. Write a program that gets configuration information of the system and shows, for example, the maximum length of arguments, the maximum number of child processes, and the maximum number of files. Check the `man` page of `sysconf(3)`.

Exercise 4. Repeat the previous exercise, but in this case for the file system configuration. For example, show the maximum number of links, the maximum size of path, and the maximum size of a filename. Check the `man` page of `pathconf(3)`.

User Information

Exercise 1. System command `id(1)` shows information about the real and effective user. Consult the `man` page and check the behaviour of the command.

Exercise 2. Write a program that shows the real and effective UID of the user. Under which circumstances could we assure that the executable file has the *setuid* bit enabled?

Exercise 3. Modify the previous program to also show the username, home directory and user information.

System Time Information

Exercise 1. The main command to show system time is `date`. Consult the `man` page and get familiar with the different available formats to show system time information.

Exercise 2. The main function to get the system time is `time(2)`. Write a program that gets the time with that functions and shows it in the terminal.

Exercise 3. The `gettimeofday(2)` function provides time information in microseconds. Write a program to measure the time spent on a loop that increments a variable a million times.

Exercise 4. Write a program that shows the current year, e.g. `"We are in 1982"`, using the `localtime(3)` function.

Exercise 5. Modify the previous program to also show the date and time in a readable way, like `"Monday, October 29 2018, 10:34"`. Use the `strftime(3)` function.

Note: To set the locale (language, time format...) in a program according to the current configuration, use function `setlocale(3)`, for example, `setlocale(LC_ALL, "")`. To set the locale in the current configuration, execute, for example, `export LC_ALL="en_US"` or `export LC_TIME="en_US"`.