Presentation

This practice proposes a series of activities with the aim the student can apply on a Unix system some of the concepts introduced in the first modules of the subject.

The student will have to do experiments and answer the proposed questions. You will also need to write a short program in C language.

The practice can be developed on any Unix system (the UOC facilitates Ubuntu 14.04 distribution). It's recommended that while you are doing the experiments there are no other users working on the system because the result of some experiments may depend on the system load.

Each question suggests a possible timing to be able to finish the practice before the deadline and the weight of the question in the final evaluation of the practice. The weight of this practice on the final qualification is 40%.

Competencies

Transversals:

- Ability to adapt to technologies and future environments by updating professional skills.

Specifics:

- Ability to analyze a problem at the level of abstraction appropriate to each situation and apply the skills and knowledge acquired to address and solve it.

- Ability to design and build computer applications using development, integration and reuse techniques.

Practice proposal

To complete this practice, we provide you with the file **pr1so.zip**, which contains source files. Unzip it using the command **unzip pr1so.zip**. To compile, for example, **prog.c**, you need to run the command **gcc -o prog prog.c** and to execute the program, you need to run **./prog**.

1. Module 2 [From October 6 to 19] (20% = 5% + 5% + 5% + 5%)

We provide the programs count1.c, count2.c and the shellscript launch.sh (you do not need to analyze how they are implemented).

1.1 Run the **uptime** command on a Unix system. Attach a screenshot of the result obtained. Interpret the meaning of the information displayed, attempting to relate it to the concepts studied in this course.

1.2 We provide you with the program **infinite1.c**; this program runs an infinite loop**.** Compile it and run it from another window. From the original window, run the **uptime** command multiple times (about 5 times in one minute). Attach the screenshots of the results shown by the uptime command. What differences do you observe compared to the screenshot from the first section? What are the reasons for these differences?

1.3 We provide you with the program **infinite2.c**, which also runs an infinite loop. Analyze its source code: how does it differ from **infinite1.c**? Compile it and run it from another window. From the original window, run the **uptime** command multiple times (about 5 times in one minute). Attach the screenshots of the results shown by the **uptime** command. What differences do you observe compared to the screenshot from the second section? What are the reasons for these differences?

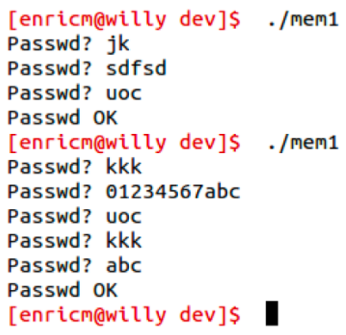
1.4 Using the ps command (properly parameterized) and filters such as **grep**, **wc**, **sort**, **cut**, **uniq**, ..., answer the following questions: (Indicate which command you would execute to answer each question).

a) How many processes are running on the machine?
b) Which process is using the most physical memory?
c) Which process has the largest virtual memory?
d) Which users have processes running?

Observation
Once you have finished this exercise, don't forget to kill all processes running the programs **infinite1.c** and **infinite2.c.**

2. Module 3: Memory [From October 20 to November 3] (70%= 20%+20%+30%)

2.1. In this activity, you will analyze the behavior of the program **mem1.c.** Study its source code. The program reads a password from the keyboard and compares it with a specific string ("uoc"). If they are the same, it displays a message indicating that the password is correct and terminates execution. If they are different, it prompts for a new password and repeats the process. We attach the result of two executions.



```
[enricm@willy dev]$  ./mem1
Passwd? jk
Passwd? sdfsd
Passwd? uoc
Passwd OK
[enricm@willy dev]$  ./mem1
Passwd? kkk
Passwd? 01234567abc
Passwd? uoc
Passwd? kkk
Passwd? abc
Passwd OK
[enricm@willy dev]$
```

Compile and run the program. Verify that it behaves as in the example. Answer the following questions:

- Why, in the second execution, is the password "abc" considered correct and "uoc" not considered correct?.
- What would need to be done to correct the error in this program?.

2.2. In this activity, you will analyze the behavior of the program **mem2.c**.

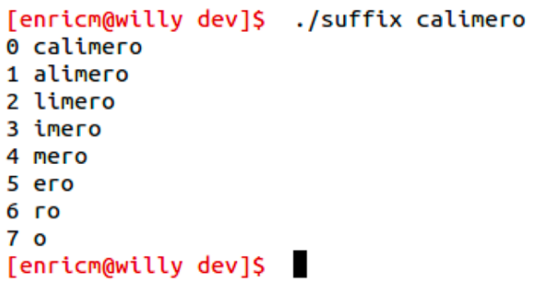
Study its source code. The program declares a series of objects: global and local variables, as well as constants. Finally, it prints the memory locations where these objects are stored. Compile and run the program.

- Attach a screenshot of the execution result.
- Why are some objects grouped in nearby memory locations? Deduce in which region of the process's logical space each object is located.

2.3 Given a string, its i-th suffix (0 < i < string length) is the string formed by the characters i, i + 1, ... string length - 1 of the string. For example, the suffixes of the string "paleta" are: paleta, aleta, leta, eta, ta, a.

Complete the code of **suffix.c** so that it fills the suffix array with all the suffixes of the string received via the argument vector (argc/argv). Once filled, the code will display the content of the array on the screen. An example of the desired result is attached:

.



```
[enricm@willy dev]$  ./suffix calimero
0 calimero
1 alimero
2 limero
3 imero
4 mero
5 ero
6 ro
7 o
[enricm@willy dev]$ █
```

Observations:

- In the program, you must use dynamic memory management routines to allocate memory both for the pointer array and to copy each suffix.
- The solution described for this problem is not the most efficient one, but it is the solution we are asking you to implement.
- Your code must be placed between the lines in suffix.c marked as /* Your code starts here */ and /* Your code ends here */.
- You are not allowed to modify the rest of the provided program.
- You must submit the source code of the program and, if it works, a screenshot showing it.

3. Module 4: In/Out [From November 4 to 10] (10%= 5%+5%)

Open two windows. In the first window, execute the command tty; this command will display a filename representing the window (for example, /dev/pts/12 or /dev/tty1). In the second window, execute the following commands:

- ps u
- ps u > xxx
- ps u > filename displayed by tty
- ps u > /dev/null

Answer the following questions:

3.1 What does each of the four commands do?
3.2 How does allowing the ps command to be executed in these four ways affect the command's programmer?

Resources

- Modules 1, 2, 3 i 4 of the subject.
- The “Operating Systems Laboratory” classroom (questions about Unix, C, …).
- Document “UNIX shell” (available in the classroom) or any others similar manual.
- Any basic language manual C.

Format and delivery date

A zip file with your campus ID will be delivered and will contain a pdf file with the answers to the questions and the source code of the program.

Delivery deadline: 24:00 on November 10, 2024.