Operating Systems (3)

Computadors – Grau en Ciència i Enginyeria de Dades

Facultat d'Informàtica de Barcelona

This Lab Session deals with the I/O subsystem, including the filesystem. This means you will get experience on managing I/O devices and how a given OS, such as Linux-based, lets processes work with them. Besides, we will analyse the performance impact of doing I/O operations following different approaches. As done in other lab sessions, use the "man" to completely understand the commands, functions and syscalls employed in this session.

Although in the following exercises we don't explicitly request the Makefile, we strongly suggest you develop and deliver one.

Download the files set from:
https://docencia.ac.upc.edu/FIB/GCED/COM/documents/Lab/S10/FilesS10.tar.gz

I/O devices

This Section aims at getting experience to identify and play with I/O devices.

Access to the "/proc/devices" file. It summarizes the different device types recognizable in the system, classified by "block" or "character" devices, and the "major" numbers (i.e. what type of device is) and the name of it.

Linux/Unix based OSes utilize a well known directory (i.e. "/dev") as the default folder to gather device files of the system. From the long listing format of the "ls" command (i.e. "ls –l") you can use the first character of the first column (i.e. "b" or "c", among others for "block" or "character" transfer based devices, respectively) as a hint to identify the device file. Just before the timestamp column, you will see two numbers separated by a comma, that represent the major and minor numbers, respectively.

Execute "ps" to show the information of the processes running in the current terminal. Among others, check the "TTY" column, that represents the terminal the process is bound to. This terminal can be found in the folder "/dev", and, in particular, in "/dev/pts". **NOTE:** *we strongly suggest to use the "bash" interpreter.*

Exercise 1

*Create and edit an **"answers.txt"** file. List the contents (using the long listing format) of the folder you have finally found the tty device of the current window you are using. Then open a new terminal window, check again the contents of the directory. Indicate in the **"answers.txt"** the tty device that the new terminal window is using, and whether the folder that gathers these terminals has the same entries or not. Indicate the major and minor numbers of the devices you find in such folder.*

Exercise 2

*Open the source code file "iochars.c", analyse the code and explain, with your own words and in the **"answers.txt"** file, what it does. Just compile and execute the binary to double check your guess.*

Exercise 3

Open another window terminal, find out the device that uses (e.g. "/dev/pts/XXX") and redirect the output of "iochars.c" to this device. ***NOTE: remember the tuple "Fd_num Permission path" (without space between "Fd_num" and "Permission") explained in the Theory lecture, where: "Fd_num" is the file descriptor to be redirected; "Permission" can be "<" (Read), ">" (Write), "<>" (Read & Write); and "path" the name of the file to be used for this redirection.*** *To validate it works as expected, you should see the characters you introduce in the source terminal written in the destination terminal. Besides, go to the "/proc/PID/fd" folder and list, using the long format, the contents of the folder to validate the summary of the "file descriptors" used by the process are the ones you expect from the explanations of Theory lectures. Write the output of this folder listing to the* ***"answers.txt"*** *file.*

Exercise 4

Copy the file "iochars.c" and name it "iochars-v2.c". Modify the code to perform exactly the same, but using a buffer (e.g. 100 chars).

Exercise 5

Execute "iochars.c" and "iochars-v2.c" redirecting the stdin (read from the file "exempleText.txt") and the stdout (to write to a new file called "sortida.dat"). Perform the execution using the following command line:

```
#>strace  -o  output.dat  -e read,write  ./programa …
```

This command line registers the trace of selected system calls (i.e. read and write) in the "output.dat" file. Execute the command line using the two binaries and compare the resulting traces. Answer in the ***"answers.txt"*** *file what differences you may find.*

Exercise 6

Open the source code files ***"ioints.c"****. It accepts a single input parameter, a number, to indicate how many integers will be written (going from values of zero to N-1). The integers will be written using integer format, rather than ASCII characters. However, the program writes to a different file descriptor, rather than the STDOUT (check the source code). In such a way, if you execute the program you will see nothing. To properly launch this program you have to redirect the right file descriptor to a given file (e.g. "output.dat"). Indicate in the* ***"answers.txt"*** *file, the required command line to perform such redirection.*

Exercise 7

Create a source code, named ***"copyFile1.c"*** *to perform the following: 1) open a file (the one you have created in the previous exercise) to read; 2) develop a loop in which every iteration it reads an integer from the file and it writes the integer, in integer format, to the "STDOUT". To properly execute this program, you have to redirect the output to a given output file, such as "CopiedOutput.dat". You can check the output file contents with "#>xxd FILE" command, showing incremental hexadecimal values, following little-endian, four-bytes each.*

Exercise 8

Copy the previous source code, and name it ***"copyFileN.c"****. Modify the code to read N numbers at a time and write N numbers at a time. The output file should have the same file size than the one obtained in the previous exercise.*

Exercise 9

Perform an experiment to execute the previous programs to read and write 1M numbers. Measure the difference in performance between "copyFile1" and "copyFileN", using "/usr/bin/time" and write the time measurements in ***"answers.txt"*** *file. If times are too low, increase the number of integers to be read and written. Check it with high values of "N", up to the maximum value possible (i.e. just to read and write the whole file using a single syscall invocation).* ***HINT:*** *You should see "copyFileN" is faster than "copyFile1" and relate the results with the number of syscall invocations, even though the difference in time could be small depending on the hardware specs. You can do a similar analysis than exercise 5 to double check this comparison.*

Upload the Deliverable

To save the changes you can use the tar command as follows:

```
#tar czvf session10.tar.gz answers.txt Makefile *.c
```

Now go to RACO and upload this recently created file to the corresponding session slot.