Exercise 1: Experiment with /proc/self

a) Execute twice the following command and explain why you get different results

```
$ ls –l /proc/self
```

a) The command “head -1” shows the first line of a text file on the standard output, execute the following command and explain the result that you get:

```
$ head -1 /proc/self/maps
```

```
$ cat /proc/self/maps | head -1
```

Exercise 2: A basic process memory map

Compile *map1.c* and run it, so do the following:

```
$ gcc -m32 map1.c –o map1
$ ./map1
```

Question 1: Analyze the memory map shown and by referring to both figure 2 and the system manual ($ man proc), answer the following questions:

1. Which is the range of logical addresses occupied by the process "*map1*"?

2. Identify the region of the code and enter its starting and ending addresses. Do the same with the stack region.

3 How many of the studied regions can you identify in the memory map of “map1” process? Identify a supported data, a non supported data region and a library code region.

4 What “map1” regions are supported by physical devices? Justify the need for such support.
5 What i-node number have the non supported regions?
6. What regions of "*map1*" may be shared by multiple processes?
7. Identify regions in table 1, where the function main and the function f are allocated

Question 2: specify in which regions, from those indicated in table 1, the system has allocated each one of the program variables.

Variable type	**Allocated region**
Initialized global variables	

Not initialized global variables:
`path_maps[80]`
`ni_glob [4095]`	

Initialized local variables	
Not initialized local variables	
Function parameters	

Exercise 3: Memory map of a process with large size local variables

Compile *map2.c* and run it, so do the following:

```
$ gcc -m32 map2.c –o map2
$ ./map2
```

Analyze the memory map shown by map2 and by referring to both figure 2 and the system manual ($ man proc), answer the following questions:

Question 3: Indicate in which region of memory is the local variable vloc.

Question 4: Compare the resulting process maps map1 and map2, and indicate if the size of their respective stacks is different. Indicate which of them is larger and justify it.

Exercise 4: Process map with dynamic allocation

Compile "map3.c" and run it, so do the following:

```
$ gcc -m32 map3.c –o map3
$ ./map3
```

The map3 process prints the memory map before and after making the dynamic memory reservation. Our aim will be compare both maps from memory, to observe what changes have happened and its relation with variable *vdin*.

Question 5: Analyze the maps of memory shown by map3 and compare them

1. Indicate whether it appears or disappears some region. Attempt to justify what these changes are due to.	
2. In the case to appear some region, what kind of region is it? What are its permissions?	
3. Indicate in which region is the contents of vector vdin.	

Exercise 5: Increasing the dynamic allocation

Work with “map3.c” code and increase the dynamic memory allocation as indicated in the following table, you have to compile and execute every time.

Question 6: Discuss what happens with the region where the variable “vdin” is allocated, try to calculate its size.

Dynamic allocation to do	**Effects on the heap region**
`malloc(1000*sizeof(int));`	
`malloc(10000*sizeof(int));`	
`malloc(20000*sizeof(int));`	
`malloc(30000*sizeof(int));`	

Exercise 6: Memory map of a process that uses libraries

Compile the program lib_cos.c, both with the static library and the implicit dynamic linking and execute it redirecting the output to a file:

```
$ gcc -m32  lib_cos.c –static –lm –o lib_static
$ gcc -m32  lib_cos.c –lm –o lib_dynamic
$ ./lib_static > libs_result
$ ./lib_dynamic > libd_result
```

Question 7: Display generated files “libs_result” and “libd_result”, compare them and answer the following questions:

1. What differences do you find when comparing the memory maps generated with the two linking types?	
2 Run the command "ls – lh” in the two executable files and compare their sizes. Try to justify the observed difference.	
3. Apply the command *size* to the resulting executable files and compare the results. Try to justify differences in size of the same regions in both executable files.	