

Gauthier Gain & Benoît Knott

# INFO0940

# OPERATING SYSTEMS

Project #2

Academic year 2023-2024



# YOUR SECOND & LAST PROJECT (1)

## You will hack the kernel

1. You will implement a Linux kernel module able to track memory usage across several processes.
2. Interaction with the kernel will occur through the pseudo file system.
3. All will be done within the kernel space. We provide a user-space program[1] in order to test your implementation.

[1] See the tester program available on [eCampus](#)

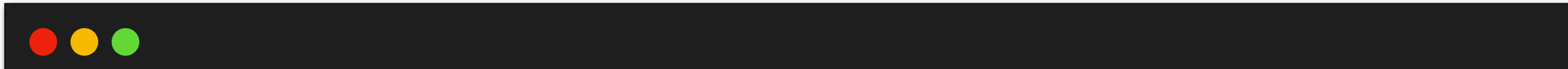
# YOUR SECOND & LAST PROJECT (2)

## It is a research project...

- ❖ This means you have to do some research yourself to understand how things work.
- ❖ It is not a waste of time. If you understand how memory works (pages, ...), you might pass the oral exam if you have a similar question.  
mainly help if question about virtualization
- ❖ It seems complicated but you will see that after some investigation, all will be clear.

# MODULE BEHAVIOUR (1)

- ❖ The module has a straightforward behaviour. During its initialization (or when the module is reset), it will create and populate an **in-memory** data structure with the names of all currently running processes on the system.
- ❖ For each set of process(es) with the same name, the module will store the following memory information:



```
int *pids; // the pid(s) of all process(es) of the set

unsigned long nb_total_pages; // the total number of pages used by all process(es) of the set

unsigned long nb_valid_pages; // the number of valid pages (i.e., pages that are present in
RAM) of all process(es) of the set

unsigned long nb_invalid_pages; // the number of invalid pages (i.e., pages that are not
present in RAM) of all process(es) of the set

unsigned long nb_shareable_pages; // the number of read-only[1] & valid pages that may be
shared (e.g., identical) between all process(es) of the set

unsigned long nb_group; // the number of groups of identical read-only[1] & valid pages.
If there are no identical pages, this field should be 0
```

[1] For simplicity.

# MODULE BEHAVIOUR (2)

new pseudofile: `memort_info`

- ❖ The module will be accessible through the pseudo file system via a file named `memory_info` in the `/proc` directory. This file will serve for both writing commands to the module and reading its output.
- ❖ Interacting with the module will be done through the following write commands:

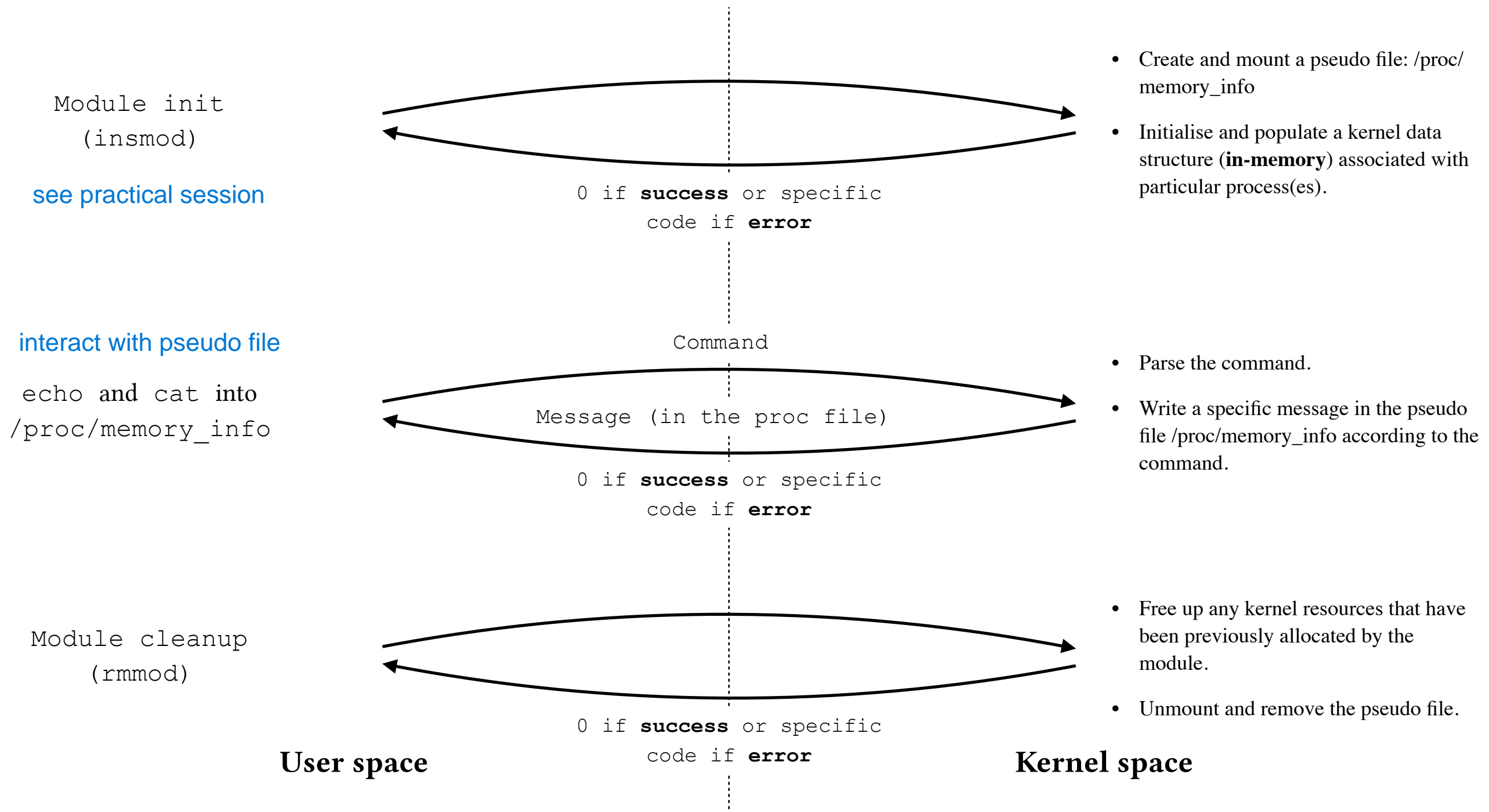
Command	Description
<b>RESET</b>	reset the in-memory data structure and populate it with the name of all currently running processes and their memory information
<b>ALL</b>	display the memory information of all processes in a specific format.
<b>FILTER   &lt;name&gt;</b>	display the memory information of all processes called “ <i>name</i> ” in a specific format (exact match).
<b>DEL   &lt;name&gt;</b>	delete the memory information of all processes called “ <i>name</i> ” (exact match)

# MODULE BEHAVIOUR (3)

- ❖ For the last two commands, the “|” character is used as a separator between the command and the name.
- ❖ Using a malformed or unknown command will be treated as an **error**.
- ❖ The module will output the result of the previous write command in the `/proc/memory_info` file.
- ❖ The output format of the `DEL` and `RESET` commands is only “[SUCCESS]” if the command is successful. Otherwise, an **error message will be displayed** (see “errors handling”).
- ❖ The output format of the `ALL` and `FILTER` commands must be as follow:

```
<name>, total: <nb_total_pages>, valid: <nb_valid_pages>, invalid: <nb_invalid_pages>,  
may_be_shared: <nb_shareable_pages>, nb_group: <nb_group>, pid(#pid): <pid1>; <pid2>; ... <pidN>
```

# HIGH-LEVEL OVERVIEW



# ERRORS HANDLING (1)

- ❖ In the kernel, it is necessary to use specific errors[1] so that users can know what causes a system/function call not to be completed.
- ❖ Depending on the error, your implementation must return an appropriate error code (**negative values**). /!\ in kernel only  
negative values
- ❖ Furthermore, your implementation must display error messages with `KERN_ERR` and print the prefix "`[ERROR]`" on the kernel log and pseudo file, followed by the error message.

[1] <https://elixir.bootlin.com/linux/v4.15/source/include/uapi/asm-generic/errno-base.h>



# ERRORS HANDLING (2)

Error	Message	Meaning
<b>ENOENT</b>	No such file or directory	The file or directory does not exist.
<b>EFAULT</b>	Bad address	The address is invalid (e.g., error during memory copy, etc.).
<b>ESRCH</b>	No such process	The process does not exist.
<b>EINVAL</b>	Invalid argument	An invalid argument was given (e.g., wrong command).
<b>ENOMEM</b>	Memory allocation error	There is not enough memory to complete the operation.

Use the message string corresponding to the associated error code.

```
ptr = kmalloc(1)
if (!ptr) {
    printf("[ERROR] memory alloc error\n");
    err = -ENOMEM;
}
```

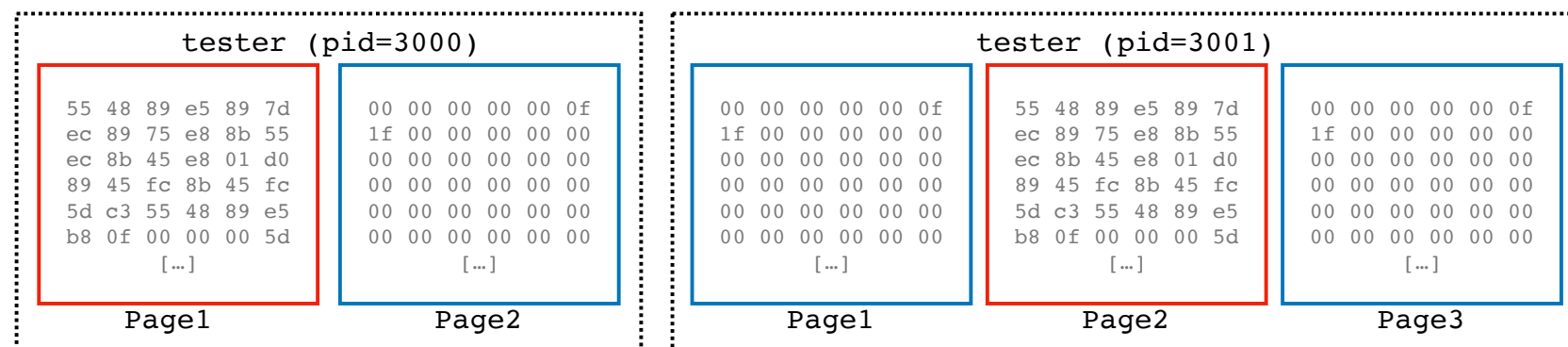
# MANAGING IDENTICAL PAGES (1)

- ❖ When multiple processes are running concurrently, it is common for certain pages of memory to be identical across these processes.
- ❖ In this assignment, you are asked to implement a mechanism to detect identical **read-only** pages between processes which have the same name and to count the number of groups of identical read-only pages (see next slide).
- ❖ Note that a single process can also have identical read-only pages.

# MANAGING IDENTICAL PAGES (2)

❖ The following figure illustrates the concept of identical pages according to two different scenarios:

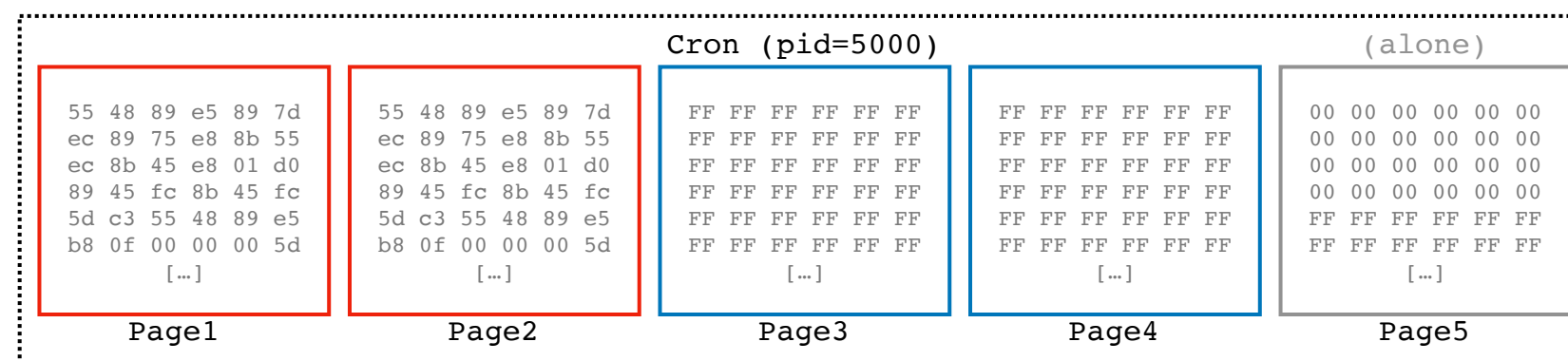
1. Scenario 1a shows a group of two processes having identical read-only pages.
2. Scenario 1b shows only one process having identical read-only pages.



Scenario 1a) The “may\_be\_shared” field is set to 5 and the “nb\_group” field is set to 2.

as 5 pages

as red, and blue forms 2 groups



Scenario 1b) The “may\_be\_shared” field is set to 4 and the “nb\_group” field is set to 2.

-> gray does not count as group neither may\_be\_shared

as red and blue forms 2 groups

# MANAGING IDENTICAL PAGES (3)

- ❖ Since this feature is the most complex and requires more research, it is marked on **6 points**.
- ❖ According to your time and your motivation, you can either implement it or skip it but in this case, you will lose **6 points**.

=> 14/20

# THE 32BITS REFERENCE MACHINE

Only use the provided reference virtual machine  
(32bits - 4.15) to test your project.

- ❖ Do not implement your module on another version of Linux than the one provided with the reference machine.
- ❖ Otherwise, your program may fail some tests on the submit platform...
- ❖ Please refer to the online [tutorial](#) to setup the required environment.

# TESTING YOUR MODULE (1)

- ❖ We provide a user-space program that can be used to see if your module can correctly detect identical pages. The program is available on eCampus.
- ❖ The program is compiled without the glibc and is statically linked.
- ❖ It allocates a specific number of **read-only & valid** pages that allow you to verify your module implementation.

```
#define NB_PAGES 10000 // The number of pages that you want (default 10000)
#define PAGE_SIZE 0x1000 // 4KB - 4096 bytes (default size of a page; DO NOT EDIT)

const int length = PAGE_SIZE * NB_PAGES;
char *addr = mmap(NULL, length, PROT, FLAGS, -1, 0);

// on-demand paging filled with 0xFF bytes
memset(addr, 0xFF, length);
// make read-only by calling the mprotect syscall
mprotect(addr, length, PROT_READ);
// infinite loop for easy tracking
while(1);
```

# TESTING YOUR MODULE (2)

- ❖ You can change the number of pages by changing the `NB_PAGES` macro.
- ❖ You can modify the previous code to create several groups of identical pages (you should call some functions multiple times).
- ❖ You can also have an idea of the total memory consumption by parsing the `smaps_rollup` file.
  - This is not an exact match but an upper bound.
  - This is only representative for **statically-linked** programs.
- ❖ The submission platform will perform basic & functional tests.

```
virt$ cat /proc/$(pidof mmap_tester)/smaps_rollup | grep "Pss:" | head -n 1
Pss:                40012 kB
# The program allocates via mmap 10000 pages of 4kB (NB_PAGES=10000)
```

# *Demonstration*



# EXAMPLE

```
# Display the memory information of all processes
echo "ALL" > /proc/memory_info && cat /proc/memory_info
sshd, total: 7845, valid: 75, invalid: 7770, may_be_shared: 4, nb_group: 2, pid(3): 861; 867; 892
cron, total: 1396, valid: 11, invalid: 1385, may_be_shared: 0, nb_group: 0, pid(1): 1752
dhclient, total: 1505, valid: 21, invalid: 1484, may_be_shared: 0, nb_group: 0, pid(1): 1711
systemd, total: 3550, valid: 420, invalid: 3130, may_be_shared: 3, nb_group: 1, pid(2): 1; 1870
zsh, total: 2065, valid: 11, invalid: 2054, may_be_shared: 0, nb_group: 0, pid(1): 1893
[...]

# Search and display the memory information of "sshd" processes
echo "FILTER|sshd" > /proc/memory_info && cat /proc/memory_info
sshd, total: 7845, valid: 75, invalid: 7770, may_be_shared: 4, nb_group: 2, pid(3): 861; 867; 892

# Search and delete the memory information of "sshd" processes
echo "DEL|sshd" > /proc/memory_info && cat /proc/memory_info
[SUCCESS]

# Does not exist anymore, so we should have an error
echo "FILTER|sshd" > /proc/memory_info && cat /proc/memory_info
[ERROR]: No such process

# Reset the data structure and populate it with the name of
# all currently running processes and their memory information
echo "RESET" > /proc/memory_info && cat /proc/memory_info
[SUCCESS]
```

# GENERAL HINTS

For this project, you need to do some research but here are some hints to help you:

- ❖ We only consider x86 (32bits) architecture.
- ❖ Do not forget to check user inputs (see session 4).
- ❖ Use printf to debug and goto for error handling (see session 4).
- ❖ Check if your kernel is upgraded to 4.15 (`uname -a`).

# HOW TO START?

- ❖ Follow the tutorial related to session 4 to have a functional kernel module manipulating the pseudo file system.
- ❖ Integrate a data structure in this kernel module, and then populate it.
- ❖ As an entry point, it will be interesting to have a look at these files:  
include/linux/sched.h and include/linux/mm\_types.h then do some googling.
- ❖ It is related to memory management so have also a look within the `mm/` subfolder (*pte*, *pgd*, etc.).
- ❖ To manage identical pages, read the code of mm/ksm.c.

# SUBMISSION

- ❖ Your submission will include your code (all the required `{.c|.h}` files in a directory named `src`) along with a `Makefile` which produces the kernel module file `module_project_os.ko`.
- ❖ As you have seen the submission platform might crash sometimes. Try to submit as soon as possible (submissions take time!).

# REQUIREMENTS

## Additional Information:

- ❖ Group of **two** that you will **keep** the whole semester (see discussion on eCampus).
- ❖ Submit a *tar.gz* archive on the submission platform (sources & report).
- ❖ You are asked to write a **very short** report (max 1 page) in which you briefly explain your implementation (see template on eCampus).
- ❖ Further information and other specifications are in the statements (available on eCampus).

Do not forget: We want clean code without warning/error.

Do not forget too: We detect **plagiarism** so do not try...

Plagiarism = **0 for the course!**

**Deadline: 17th May 2024**

*Happy Coding!*