PC 5

System call

Exercice

- Clone the git repository
  git clone http://gitlab.montefiore.ulg.ac.be/INFO0940/kernel-4.4.50.git
- Make a "PC4" branch
- Add a sys_forkexec system call
  - It is the equivalent of calling fork
    - followed by exec in the child
    - while the parent returns directly
      - Bonus : the parent waits for the child
  - You're supposed to copy paste fork and exec content, using all default parameters like your shell did
- Compile, install, reboot, try
- Make a program that uses sys_forkexec to call /bin/ls (no shell, don't mess it up)
- Go back to the master branch for the project 4 !

Help yourself with last week's slides "Progamming in the Linux Kernel"

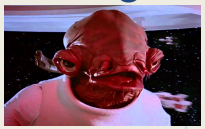
Hint #1

Define a syscall

How to add a system call

- Search the kernel source for existing syscall definition and reference like sys_mkdir which has two arguments too
- you must declare your handler (the real function which does something for the syscall) with

asmlinkage int sys_{syscallname}({args}) or a macro which does it, depending of what you think is best



NOTE:

- user-space entries have to be double-checked ! A BSOD cannot occur on a modern computer, you can’t say “It’s the user which misused my syscall !”
- new syscalls always affect the *core* kernel, and can't be defined simply through a *module*.

Hint #2

Use a syscall

Using a syscall in a C program

```
#include <sys/syscall.h>

...

int main() {
        ...
        int result =  syscall(359, arg1, arg2);
        ...
}
```

C Code

where 359 is the number chosen for your syscall. Result is the value your syscall will return.

More info? “man syscall”.
Without knowing it, when programming you call functions which call syscall…
You use it all the time !

Project 4

Deadline Thursday 29/03 23h59

System call

- You will create a new system call that will return statistics about virtual memory page faults for one or multiple given process.

- Create a new system call named “sys_pfstat” that takes as argument a PID and a struct pfstat*.

- It will
    - A) set the the process in "pfstat mode"
    - B) Recover the given process's statistics if it was already in pfstat mode, put them in the userlevel structure passed as argument and reset the process statistics to 0.

struct pfstat

```
struct pfstat {
	int stack_low; //Number of times the stack was expanded after a page fault
	int transparent_hugepage_fault; //Number of huge page transparent PMD fault
	int anonymous_fault; //Normal anonymous page fault
	int file_fault; //Normal file-backed page fault
	int swapped_back; //Number of fault that produced a read-from swap to put back the page online
	int copy_on_write; //Number of fault which backed a copy-on-write;
	int fault_alloced_page; //Number of normal pages allocated due to a page fault (no matter the code path if it was for an anonymous fault, a cow, ...).
}
//This is subject to interpretations ! Justify in the report
```

sys_pfstat(pid_t pid, struct pfstat* pfstat)

- If pid is not a valid PID, return an error code of 1
- If the pfstat ptr is not valid, return an error code of 2
- If an error should occur, a negative error value should be returned, and an information about the error printed with printk, beginning with “[PFSTAT] Error : ”
- If everything is ok, its return value is 0
- If the arguments are valid, it will print the message “[PFSTAT] Process %s is now in PFSTAT mode” where %s is the process name. If it was in PFSTAT mode the message will not be printed.
- When the syscall is entered, print the line
  - [PFSTAT] Syscall entered!

Shell

- Update your shell to support
  - *command &* to launch the command in background
    - When it succeeds, as there is no return code (yet) you must show the "> " prompt, as when the shell first starts
  - $! returns now the **PID** of the last process executed **in background**
  - sys pfstat PID
    - Use the syscall to prints every stats variable of pfstat in order in the format "variable_name VALUE\n"

Shell example

The idea is to execute the following command :

```
> ./my_test_program &
> pid=$!
0> sys pfstat $pid
stack_low 0
…
fault_alloced_page 0
0> sleep 5
0> sys pfstat $pid
stack_low 7
..
fault_alloced_page 78
```

Obligations

- Respect the exact strings for printk.
- Use tools/library provided by the kernel when applicable. If you make a linked list, use klist, do not implement a RB-Tree yourself, …
- When you “pick a number” you use the first available
- Stay on my virtual image, with the 32bit ubuntu OS, and one virtual processor. **But I will never ask you the image, do what you want on it.**
- Do nothing for any other arch that x86 32bit, so I’m sure you did not add things randomly. This include x86_64 specific code !
- Choose wisely where you implement the sys_pfstat code.

Report

- **Follow the same rules than for previous steps, including steps 3 (comments, patch, report, submission, …)**
- Explain what you did, where, how why, how you find it.
- Why you updated the counters at the place you choose.
- The problems you had
- How you solved them
- Comment each line added/changed regarding the system call definition, what it does (only definition, not the implementation of sys_pfstat itself).

Submission

- A tar.gz
  - A Makefile to build the shell as "shell" executable
    - The source must not depend on the syscall kernel headers, ie use the generic "syscall" function with a hardwired number and struct pfstat definition.
  - A report
  - A patch
- **Follow the same rules than for previous steps including step 03 (comments, patch, report, submission, …)**
  - **http://www.tombarbette.be/courses/os/**
  - **Your are now at step 04 → g42s04-source.patch**
  - **Archive name, report name are defined also**