Project #1

Implement a simple shell in C

- You will implement a simple shell, which uses exclusively **execv** and **fork** for sub-process.
  - Fork duplicates the process
  - Execv replace the current process by another one

- It must support launching programs with arguments, and return to the prompt when the program terminates, but no other concepts such as variable, substitution, pipes, chaining, …

- It will exit upon typing "exit" or CTRL+D (EOF)

- Command line can be limited to 255 arguments, while each arguments may be limited to 255 characters.

- The first prompt must be "> ",then "RET> " where RET is the return code of the last command. If there was no command when pressing enter, "> " is shown

Example

```
$ gcc -std=gnu99 -o shell shell.c && ./shell
> /bin/ls
shell  shell.c       shell.c~  shell.tar.gz

0> /bin/ls -al
total 32
drwxr-xr-x. 2 tom tom 4096  9 fév 12:23 .
drwxr-xr-x. 3 tom tom 4096  9 fév 11:28 ..
-rwxr-xr-x. 1 tom tom 9032  9 fév 12:23 shell
-rw-r--r--. 1 tom tom 1133  9 fév 12:23 shell.c
-rw-r--r--. 1 tom tom 1134  9 fév 12:23 shell.c~
-rw-r--r--. 1 tom tom  208  9 fév 11:41 shell.tar.gz

0>
```

Terminal

PATH support

- Look in the folders of $PATH (separated by :) in order to find the program to launch
- This should avoid to type "/bin/ls", as bin should be in the path, typing ls should be enough

DO NOT USE exec*p variants that do that themselves, only support it manually

Project 2 (08/03 23h59)

- You will add a "sys" built-in to your shell
  - sys hostname → Gives the hostname without using a system call
  - sys cpu model→ Gives the CPU model
  - sys cpu freq N → Gives the CPU number N frequency
  - sys cpu freq N X -> Set the frequency of the CPU N to X (in HZ)
    - Prints nothing
  - sys ip addr DEV -> Get the ip and mask of the interface DEV
    - a.b.c.d e.f.g.h
  - sys ip addr DEV IP MASK -> Set the ip of the interface DEV to IP/MASK
- Built-in must return error code like real software
- Support variables along with $? and $! replacement

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 value 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
```