HW1: programming exercise

This assignment practices basics of UNIX programming and system calls. Your program should consist of a *main* and the functions detailed below.

1)

Write the function ***pid_t my_fork (void)***, which does the followings:
- Call fork()
- If fork() succeeds (namely, its return value is non-negative), return the value returned by fork().
- Else (namely, fork() failed, from any reason), print an error message, and exit the process.

2)

Write the function: ***void print_pids (int fd, short unsigned int N, short unsigned int G)***, where

For each *g=0, 1, ..., G,* the function should print N^g lines in the following format:
My pid is p. My generation is g.
Where:
- *g* is the *generation* of the process: the generation of the original (first) process is 0. If the generation of a process is *i*, then the generation of its child process is *i+1*.
- *p* is the pid of the process which prints the line.

For instance, the line may be:
My pid is 1038. My generation is 0.

Note that the pids are arbitrary, and are depended upon the processes' scheduling at your machine. In particular, the operating system may allocate a new process the same pid of an old terminated process.
The function should print its output to the file, whose file descriptor is the input *fd*.

Limitations and hints:
- The function should use **a single loop**.
- Each process iterates exactly ***N* times** over the loop.
- Each process prints a single line.
- Use *my_fork()*, not *fork()*.
- No prints neither calls to *my_fork()* are allowed outside the loop.
- Each process should print its line only **after all its children printed their lines.**

3)

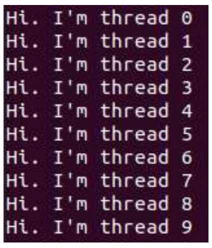
Write the function: **void count_lines (short unsigned int G)**.
This function counts the number of lines in the file *out.txt* (to which we will write output, as described below); and then prints **to the screen** output which follows the following format:
Number of lines by processes of generation 2 is 9
Number of lines by processes of generation 1 is 3
Number of lines by processes of generation 0 is 1

Limitations and hints:
- The function should use **a single loop**.
- The function should generate either *G-1* or *G* new processes.

- Each process should print a single line: the line referring to generation *g* should be printed by a process of generation *g*.
- You may use the commands *system()*, *grep* and *wc*.
- The output should be written in **decreasing** generation order (as in the sample above).

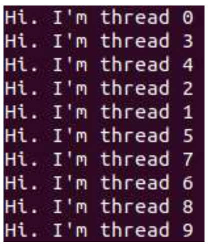
4. Write the function: **void print_threads(*short unsigned int N*).** *The function should create N threads such that each thread print "* **Hi. I'm thread number i"** where i is the i-th thread that created. The order of the prints **MUST** appear in the right order.
For example:

Good code:



```
Hi. I'm thread 0
Hi. I'm thread 1
Hi. I'm thread 2
Hi. I'm thread 3
Hi. I'm thread 4
Hi. I'm thread 5
Hi. I'm thread 6
Hi. I'm thread 7
Hi. I'm thread 8
Hi. I'm thread 9
```

Bad code:



```
Hi. I'm thread 0
Hi. I'm thread 3
Hi. I'm thread 4
Hi. I'm thread 2
Hi. I'm thread 1
Hi. I'm thread 5
Hi. I'm thread 7
Hi. I'm thread 6
Hi. I'm thread 8
Hi. I'm thread 9
```

5. Write the main function: **int main (int argc, char* argv[])**, which:
* opens a file named *out.txt* (for simplicity, we'll use always the same output file name). If the file already exists, its content will be re-written by the program. Else, it will be created.
* Calls *print_pids* with the descriptor of *out.txt* and with argv[1], argv[2] as N, G respectively.
* Calls *count_lines()* to check the results of *print_pids()*.
* Calls print_threads(N).

6. Write a makefile which compiles the code to an executable named *OS*.

Example
Running the following lines
> make
> ./OS 3 2
Should:

- Generate the file named *out.txt* which you'll have in Moodle.
- print to the screen the following lines:
  Number of lines by processes of generation 2 is 9
  Number of lines by processes of generation 1 is 3
  Number of lines by processes of generation 0 is 1
- *print to the screen the following lines:*
  Hi. I'm thread number 0
  Hi. I'm thread number 1
  Hi. I'm thread number 2

You may assume that the input to *main* is correct.
The HW should accurately follow the submission instructions in the course's site.