B2- Synthesis Pool

B-ADM-144

Project Tester

automated project tester draft

v1.2

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Requirement

my_ps_synthesis

Write a c file named `requirement.c` that contains a fork function. The function should be prototyped the following way:

```
void my_ps_synthesis();
```

In the parent process, call the `wait` function.
In the child process, execute the `ps` command.
Test your program and assess that the execution of the father is stopped.

The entire libC is allowed, except the `system` function.
The rest of the project will not be corrected unless this requirement is fully functional (and rewritten).

The file must be placed at the root of your git repository.
It will be compiled with our main function, and our Makefile (the `-I` flag being empty).

Project Tester

automated project tester draft

binary name:	projTester
repository name:	ADM_projTester_$YEAR
repository rights:	ramassage-tek
language:	C
group size:	1
compilation:	via Makefile, including re, clean and fclean rules

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Authorized functions: every functions from the libC, except 'system'

The goal of this project is to automate the process of tests execution.

All the tests to be executed will be organized in the following way:

1. a tests root directory (TRD), given as argument to your program, and containing the whole set of tests (grouped via subdirectories or not),
2. TRD sub-directories (or sub-sub-directories, or...) containing a group of tests (the name of the directory being the name of the group of tests),
3. files called Test Description Files (TDFs) that describes tests (the name of the test being the name of the associated file). The extension of those files must be **.tdf**. Their content will be clarified further on in this document.

Given a TRD and a binary file to test (BFT), the objective of your program is to execute all the tests on the BFT, and output results.

1- Searching the TRD

First, no BFT will be given to your program.
In such a case, you have to output the tree structure of the text.

The files and directories must be sorted alphabetically.

Terminal

```
~/B-ADM-144> ./projTester ./rootDirectory1
rootDirectory1/
-----option1/
----------client/
---------------test1.tdf
----------killerTest.tdf
----------server/
---------------test1.tdf
---------------test2.tdf
-----option2/
----------bigTest.tdf
----------smallTest.tdf
-----rigor/
----------noArg.tdf
----------tooManyArgs.tdf
-----rootTest.tdf
```

2- BFT

The BFT is now given as argument to your program. Your program must search for the BFT, first in the current directory, then in the $PATH.
If the binary is not found, quit cleanly with an error message.

Remember, when your program exit with an error, it should have an exit status of 84.

3- Arguments

Initially, the TDFs will only contain one line formatted the following way: *ARGS:arg1 arg2....*
For each test to be executed, your program has to execute the BFT with the parsed arguments, and print the corresponding output.

The line '*ARGS:*" corresponds to the test without argument.

Terminal

```
~/B-ADM-144> cat ./rootDirectory1/rigor/tooManyArgs.tdf
ARGS:arg1 arg2 arg3
```

Terminal

```
~/B-ADM-144> ./projTester ./rootDirectory1 echo
[option1] [client] test1: 2 3
[option1] killerTest: /dev/random 42
[option1] [server] test1: 4 3
[option1] [server] test2: 4 12
[option2] bigTest: 120004 423455
[option2] smallTest: 1 2
[rigor] noArg:
[rigor] tooManyArgs: arg1 arg2 arg3
rootTest: 34 23
```

The format of the output must be the same as this one for now on.

4- Input

A new line is now added to the TDFs, in order to redirect the standard input. The syntax of this line is as follows: *INPUT:file*.
This line is optional.

Terminal

```
~/B-ADM-144> ./projTester ./rootDirectory2
rootDirectory2/
-----loneTest.tdf
-----toto
```

Terminal

```
~/B-ADM-144> cat ./rootDirectory2/loneTest.tdf
ARGS:-e
INPUT:toto
```

Terminal

```
~/B-ADM-144> cat ./rootDirectory2/toto
First line
Second line
```

Terminal

```
~/B-ADM-144> ./projTester ./rootDirectory2 cat
loneTest: First line$
Second line$
```

5- Output

Add the possibilty of printing the results into an output file to your program.
This file will replace the standard output in order to print the results.

If the file doesn't exist, it will be created. Otherwise, the output of your progam will be concatenated at the end of the file.

Terminal

```
~/B-ADM-144> ./projTester TRD [BFT] [outputFile]
USAGE
          TRD        root directory of all the tests
          BFT        binary file to be tested
          outputFile file in which the ressult is printed
```

6- Awaited result

The TDFs can now contain a line giving the awaited test result.
This line is optional, and will be formatted the following way: *RES:string*.
If the output of the BFT **contains** the corresponding string, which can be a regexp, your program will print *OK*. Otherwise, it will print *KO*.

Terminal

```
~/B-ADM-144> ./projTester ./rootDirectory1 echo toto
~/B-ADM-144> cat toto
[option1] [client] test1: OK
[option1] killerTest: OK
[option1] [server] test1: KO
[option1] [server] test2: KO
[option2] bigTest: KO
[option2] smallTest: KO
[rigor] noArg: OK
[rigor] tooManyArgs: OK
rootTest: OK
```