TP3: Fork and pipes

Offspring and application to launch a program

Look at the functions `fork`, `execvp` and `wait`.

a. Write a program that creates a second process and prints "I am the father" or "I am the child".

b. Write a program that executes the command "ls" and writes "ls is done" when the execution of the program is done.

c. Implement a function `int launch_a_program(char* name, char** args)` in your mini-shell that executes this program and test it.

A Cro-Magnon way to communicate : sending signals

Two processes can communicate with each other using signals. If you want to send a signal to a process, you can use the command `kill`.

a. Look at the manual of `sigaction` and find how can a program react to a signal.

b. Test it by creating a program such has when you type "kill the_pid_of_my_program", it writes "I dont want to die".

c. Application : when you type control-C in your mini-shell, it sends the corresponding signal to the program that is currently executed.

Communication 2 : using pipes

A pipe is a way of communication between two processes. A way to use it is the following way :

```
pipe(tube);
pid = fork();
if (pid == 0) {
  dup2(...);
  close(...);
  close(...);
  ...
}
else
{
  dup2(...);
  close(...);
  close(...);
  ...
}
```

a. Explain why the call of the function `pipe()` is done before the call of the function `fork()`.

b. Using this pipe, we want to create a program that forks and and redirects the standard output of the father to the standard input of its child. Complete the call of the function `dup2()` and `close()`.

c. Test your program by creating a father that writes to its child some lines and a child that prints "my father says xxx".

d. Create a program that executes `ls | grep xxx`.

e. Integrate the use of the pipes in your shell. At first you can deal with only two commands in the shell, without arguments nor redirections (>).

f. Try to reduce some of these limitations.

g. Implement the functions > and `&`.