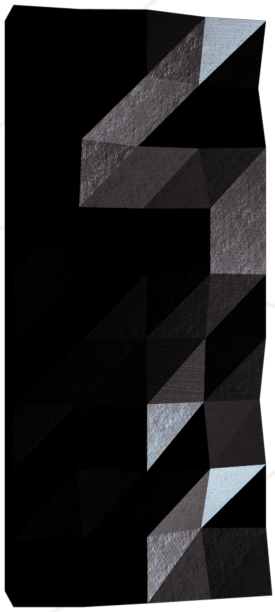

Pipex

Summary: This project is the discovery in detail and by programming of a UNIX mechanism that you already know.

Version: 2

Contents

I Foreword **2**

II Common Instructions **3**

III Objectives **5**

III.1 Examples 5

IV Bonus part **6**

V Submission and peer correction **7**

Chapter I

Foreword

Cristina: `"Go dance salsa somewhere :)"`

Chapter II

Common Instructions

- Your project must be written in C.
- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a `0` if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, use cc, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

- The executable file must be named `pipex`.
- You have to handle errors sensitively. In no way can your program quit unexpectedly (Segmentation fault, bus error, double free, etc). If you are unsure, handle the errors like the shell command `< file1 cmd1 | cmd2 > file2`.
- Your program cannot have memory leaks.
- You are allowed to use the following functions:
  - `access`
  - `open`
  - `unlink`
  - `close`
  - `read`
  - `write`
  - `malloc`
  - `waitpid`
  - `wait`
  - `free`
  - `pipe`
  - `dup`
  - `dup2`
  - `execve`
  - `fork`
  - `perror`
  - `strerror`
  - `exit`

Chapter III

Objectives

Your objective is to code the Pipex program.
It should be executed in this way:

```
$> ./pipex file1 cmd1 cmd2 file2
```

Just in case: file1 and file2 are file names, cmd1 and cmd2 are shell commands with their parameters.

The execution of the pipex program should do the same as the next shell command:

```
$> < file1 cmd1 | cmd2 > file2
```

III.1 Examples

```
$> ./pipex infile ``ls -l'' ``wc -l'' outfile
```

should be the same as "< infile ls -l | wc -l > outfile"

```
$> ./pipex infile ``grep a1'' ``wc -w'' outfile
```

should be the same as "< infile grep a1 | wc -w > outfile"

Chapter IV

Bonus part

Bonuses will be evaluated only if your mandatory part is PERFECT. By PERFECT we naturally mean that it needs to be complete, that it cannot fail, even in cases of nasty mistakes like wrong uses, etc. It means that if your mandatory part does not obtain ALL the points during the grading, your bonuses will be entirely IGNORED.

- Handle multiple pipes :

```
$> ./pipex file1 cmd1 cmd2 cmd3 ... cmdn file2
```

Must be equivalent to :

```
< file1 cmd1 | cmd2 | cmd3 ... | cmdn > file2
```

- Support « and » when the first parameter is "here_doc"

```
$> ./pipex here_doc LIMITER cmd cmd1 file
```

Must be equivalent to:

```
cmd << LIMITER | cmd1 >> file
```

Chapter V

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.