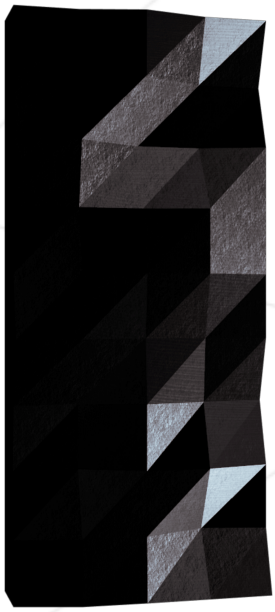

Pipex

Summary: This project will allow you to explore a UNIX mechanism in detail, one that you are already familiar with, by implementing it in your program.

Version: 3.2

Contents

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.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc.) except for undefined behavior. If this occurs, your project will be considered non-functional and will receive a `0` during the evaluation.
- All heap-allocated memory must be properly freed when necessary. Memory leaks will not be tolerated.
- If the subject requires it, you must submit a `Makefile` that compiles your source files to the required output with the flags `-Wall`, `-Wextra`, and `-Werror`, using `cc`. Additionally, your `Makefile` must not perform unnecessary relinking.
- Your `Makefile` must at contain at least the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To submit bonuses for your project, you must include a `bonus` rule in your `Makefile`, which will add all the various headers, libraries, or functions that are not allowed in the main part of the project. Bonuses must be placed in `_bonus.{c/h}` files, unless the subject specifies otherwise. The evaluation of mandatory and bonus parts is conducted separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` into a `libft` folder. 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 **does not need to be submitted and will not be graded**. It will give you an opportunity to easily test your work and your peers' work. You will find these 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 the assigned Git repository. Only the work in the Git repository will be graded. If Deepthought is assigned to grade your work, it will occur

after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Mandatory part

Program name	`pipex`
Turn in files	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`file1 cmd1 cmd2 file2`
External functs.	• `open, close, read, write, malloc, free, perror, strerror, access, dup, dup2, execve, exit, fork, pipe, unlink, wait, waitpid` • `ft_printf or any equivalent YOU coded`
Libft authorized	`Yes`
Description	`This project focuses on handling pipes.`

Your program should be executed as follows:

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

It must take 4 arguments:

- `file1` and `file2` are **file names**.
- `cmd1` and `cmd2` are **shell commands** with their parameters.

It must behave exactly like the following shell command:

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

III.1 Examples

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

Its behavior should be equivalent to: `< infile ls -l | wc -l > outfile`

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

Its behavior should be equivalent to: `< infile grep a1 | wc -w > outfile`

III.2 Requirements

Your project must comply with the following rules:

- You must submit a `Makefile` that compiles your source files. It must not perform unnecessary relinking.
- Your program must never terminate unexpectedly (e.g., segmentation fault, bus error, double free, etc.).
- Your program must not have **memory leaks**.
- If you are unsure, handle errors the same way as the shell command: `< file1 cmd1 | cmd2 > file2`

Chapter IV

Bonus part

You can earn extra points if you:

- Handle multiple pipes.

  This:

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

  Should behave like:

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

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

  This:

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

  Should behave like:

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

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

Submission and peer-evaluation

Submit your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

file.bfe:VACsSfsWN1cy33ROeASsmsgnY0o0sDMJev7zFHhw
QS8mvM8V5xQQpLc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZ0