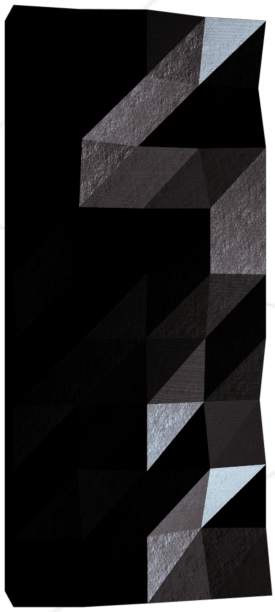

Minitalk

Summary:
The purpose of this project is to code a small data exchange program using UNIX signals.

Version: 2

Contents

Chapter I

Foreword

The cis-3-Hexen-1-ol, also known as (Z)-3-hexen-1-ol and leaf alcohol, is a colorless oily liquid with an intense grassy-green odor of freshly cut green grass and leaves.

It is produced in small amounts by most plants and it acts as an attractant to many predatory insects. cis-3-Hexen-1-ol is a very important aroma compound that is used in fruit and vegetable flavors and in perfumes.

The yearly production is about 30 tonnes.

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}` if the subject does not specify anything else. 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.

Chapter III

Project instructions

- Name your executable files `client` and `server`.
- You have to turn in a `Makefile` which will compile your source files. It must not relink.
- You can definitely use your `libft`.
- You have to handle errors thoroughly. In no way your program should quit unexpectedly (segmentation fault, bus error, double free, and so forth).
- Your program mustn't have **memory leaks**.
- You can have **one global variable per program** (one for the client and one for the server), but you will have to justify their use.
- In order to complete the mandatory part, you are allowed to use the following functions:
  - `write`
  - `ft_printf` and any equivalent YOU coded
  - `signal` -- simplified software signal facilities
  - `sigemptyset` -- manipulate signal sets
  - `sigaddset` -- manipulate signal sets
  - `sigaction` -- software signal facilities
  - `kill` -- send signal to a process
  - `getpid` -- get parent or calling process identification
  - `malloc`
  - `free`
  - `pause` -- stop until signal
  - `sleep` -- suspend execution for an interval of time
  - `usleep` -- suspend thread execution for an interval measured in microseconds
  - `exit` -- perform normal program termination

Chapter IV

Mandatory Part

You must create a communication program in the form of a **client** and a **server**.

- The server must be started first. After its launch, it has to print its `PID`.
- The client takes two parameters:
  - The server `PID`.
  - The string to send.
- The client must send the string passed as a parameter to the server.
  Once the string has been received, the server must print it.
- The server has to display the string pretty quickly. Quickly means that if you think it takes too long, then it is probably too long.

`1 second for displaying 100 characters is way too much!`

- Your server should be able to receive strings from several clients in a row without needing to restart.
- The communication between your client and your server has to be done **only** using UNIX signals.
- You can only use these two signals: `SIGUSR1` and `SIGUSR2`.

`Linux system does NOT queue signals when you already have pending signals of this type! Bonus time?`

Chapter V

Bonus part

Bonus list:

- The server acknowledges every message received by sending back a signal to the client.
- Unicode characters support!

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 VI

Submission and peer-evaluation

Turn in 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.