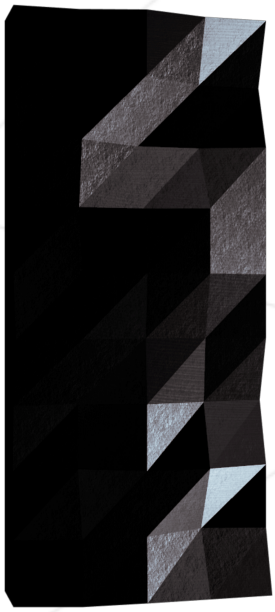

Minitalk

Résumé:
Ce projet a pour but de vous faire réaliser un petit programme d'échange de données utilisant les signaux UNIX.

Version: 3

Table des matières

I	**Préambule**	**2**
II	**Règles communes**	**3**
III	**Consignes spécifiques au projet**	**4**
IV	**Partie obligatoire**	**5**
V	**Partie bonus**	**6**
VI	**Rendu et peer-evaluation**	**7**

Chapitre I

Préambule

Voici les paroles du générique de *Firefly* :

```
Take my love, take my land
Take me where I cannot stand
I don't care, I'm still free
You can't take the sky from me.

Take me out to the black
Tell them I ain't comin' back
Burn the land and boil the sea
You can't take the sky from me.

Leave the men where they lay
They'll never see another day
Lost my soul, lost my dream
You can't take the sky from me.

I feel the black reaching out
I hear its song without a doubt
I still hear and I still see
That you can't take the sky from me.

Lost my love, lost my land
Lost the last place I could stand
There's no place I can be
Since I've found Serenity

And you can't take the sky from me.
```

Ce projet est plus facile si vous avez regardé l'intégralité de *Firefly*.

Chapitre II

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions ne doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier différent : `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre III

Consignes spécifiques au projet

- Les fichiers exécutables doivent être nommés `client` et `server`.
- Vous devez rendre un `Makefile` qui compilera vos fichiers sources. Il ne doit pas `relink`.
- Vous devez gérer les erreurs avec du bon sens. En aucun cas votre programme ne doit quitter de manière inattendue (faute de segmentation, erreur de bus, double free, etc.).
- Votre programme ne doit pas avoir de **fuites de mémoire**.
- Vous pouvez utiliser **une variable globale par programme** (une pour le client et une pour le serveur) mais leur usage doit être justifié.
- Afin de faire la partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes :
  - `write`
  - `ft_printf` et tout équivalent que VOUS avez codé
  - `signal`
  - `sigemptyset`
  - `sigaddset`
  - `sigaction`
  - `kill`
  - `getpid`
  - `malloc`
  - `free`
  - `pause`
  - `sleep`
  - `usleep`
  - `exit`

Chapitre IV

Partie obligatoire

Vous devez réaliser un programme de communication sous la forme d'un **client** et d'un **serveur**.

- Le serveur doit être lancé en premier et doit, après le lancement, afficher son `PID`.
- Le client prend deux paramètres : et aussi l envoyer comme argument au client
  - Le `PID` du serveur.
  - Une chaîne de caractères à transmettre.
- Le client doit communiquer au serveur la chaîne passée en paramètre.
  Une fois la chaîne entièrement reçue, le serveur doit l'afficher.
- Le serveur doit être capable d'afficher la chaîne rapidement. Par rapidement, on entend que si vous pensez que c'est trop long, alors c'est sûrement trop long.

`1 seconde pour afficher 100 caractères, c'est COLOSSAL !`

- Votre serveur doit pouvoir recevoir des chaînes de plusieurs clients à la suite sans nécessiter d'être relancé.
- La communication entre vos programmes doit se faire **uniquement** à l'aide de signaux UNIX.
- Vous ne pouvez utiliser que les deux signaux suivants : `SIGUSR1` et `SIGUSR2`.

`Le système Linux ne met PAS les signaux en file d'attente lorsque vous avez déjà un signal en attente de ce type ! Bonus ?`

Chapitre V

Partie bonus

Liste des bonus :

- Le serveur confirme la réception de chaque message en envoyant un signal au client.
- Le support des caractères Unicode !

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Chapitre VI

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

file.bfe:VAAe8ElCrUAbXivzOueiIpv/u/ia9PL50+HI+8/bgPKLESHlp
tPLpuOPW9zWV/LwDVaOqCRCGu6Gopk1XOi6Kn7t