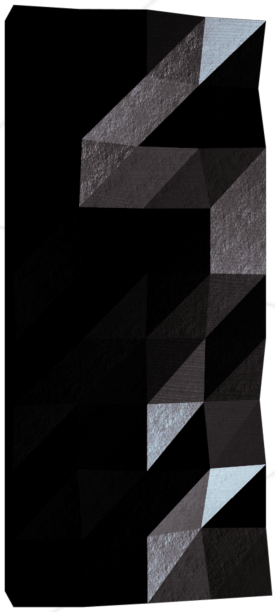

Minishell

Beau comme un shell

Résumé: Ce projet a pour but de vous faire faire un début de shell tout ce qu'il y a de plus simple. Le Shell c'est beau ! D'ailleurs ne dit-on pas : "beau comme un shell ?". Vous allez, grâce à l'ensemble des projets shell approcher de l'insondable puissance de l'intelligence de l'Homme (je sais même pas si ça vaut le coup de vous le faire découvrir, mais on m'a forcé alors bon...).

Table des matières

Chapitre I

Préambule

I.1 Le codeur, le samouraï des temps modernes !

En vrai je n'ai aucune idée de comment ça ce passait à l'époque des samouraï alors la comparaison n'est pas forcément totalement juste.

Nous aborderons quelques notions de Kendo au cours de ces shell (il y en a plusieurs), parce que le Kendo est important dans la pratique de la programmation.

À la base, le Kendo est un sport qui a été conçu dans le cadre d'un entrainement au combat à l'épée dans le Japon médiéval. L'idée de base est que quand vous perdez, vous êtes... mort[1]. Aussi, il est vivement conseillé de s'entrainer d'arrache-pied pour réussir à survivre et comprendre quelques notions, en particulier de la discipline.

Là où ça rejoint donc la programmation, c'est que nous sommes les Samouraï des temps modernes. Une erreur de notre part et nous en avons pour nos frais. Inutile de vous rapeller que désormais, une opération chirugicale ne se fait pas sans utiliser de l'électronique et nous sommes ceux qui allons coder ce système.

Aussi commençons par ce fameux crash d'Ariane 5 en 1996, qui ne valait pas 19,999999999 sur 20 mais bien 0 et le tout à cause d'un bit (pas un byte, je parle bien de 1 bit).

1. Et tous le monde sait qu'il ni a rien de plus chiant que de se faire tuer.

I.2 Vol 501 d'Ariane 5, ou " l'éloge de la programation bien faite"

Ceci est une copie de la page Wikipedia mais compte tenu de votre niveau déplorable, en cliquant sur un lien vous risquez de vous tromper, donc je recopie.

Vous allez vous tromper !

Le vol 501, vol inaugural de la fusée Ariane 5, a eu lieu le 4 juin 1996 et s'est soldé par un échec. La fusée s'est brisée et a explosé en vol, 40 secondes après le décollage, à la suite d'une panne du système de navigation. L'incident, dû à un bug dans les appareils informatiques utilisés par le pilote automatique, a provoqué la destruction de la fusée ainsi que la charge utile – 4 sondes de la mission Cluster – d'une valeur totale de 370 millions de dollars, ce qui en fait le bug informatique le plus coûteux de l'histoire. La fusée a explosé à une altitude de 4000 mètres au-dessus du centre spatial de Kourou, en Guyane française. L'incident n'a fait aucune victime (la fusée n'était pas habitée).

I.2.1 Chronologie des événements

Le lancement a lieu le 4 juin 1996 à 9h35, avec une heure de retard en raison de mauvaises conditions météo. Il s'agit du premier lancement de la fusée Ariane 5. À 9h35, les moteurs de la fusée se mettent en route, et les deux systèmes de guidage inertiel (le système principal et de secours) commencent à mesurer les mouvements de rotation et d'accélération de la fusée. Après 37 secondes de vol, les fortes accélérations de la fusée provoquent un dépassement de capacité dans le calculateur du système de guidage inertiel principal, qui se met aussitôt hors service. Le système de guidage de secours (identique à l'autre) subit la même avarie, et s'arrête à la même seconde. Le pilote automatique, qui s'appuie sur les informations provenant du système de guidage inertiel, n'a alors plus aucun moyen de contrôler la fusée. L'échec de la mission est inéluctable. 3 secondes plus tard, le pilote automatique se met en route. Suite à une mauvaise interprétation du signal de panne des deux guidages inertiels hors service, le pilote automatique provoque une violente correction de trajectoire. Les tuyères des boosters et du moteur principal sont tournées jusqu'en butée, et la fusée part en virage serré. La fusée dérape de sa trajectoire, et les boosters sont arrachés par le courant d'air décentré. Ce qui déclenche le mécanisme d'auto-destruction préventive de la fusée. Le contrôleur de vol au sol, ayant perdu tout contact avec la fusée, télécommande son auto-destruction, cependant la fusée a déjà explosé. Les débris de la fusée, qui se trouvait à environ 4000 m d'altitude, sont projetés au loin et s'éparpillent sur une surface d'environ 12 km2 aux abords du centre spatial de Kourou, en Guyane française.

I.2.2 Causes

Un système de guidage inertiel est un ensemble composé d'un appareil informatique, d'accéléromètres et de gyroscopes qui permettent de mesurer les mouvements effectués par un véhicule. L'appareil informatique calcule la position, la vitesse et l'inclinaison du

véhicule sur la base des mesures d'accélération et de rotation obtenues par les capteurs. C'est un équipement standard dans les bateaux, les avions, les missiles et les véhicules spatiaux. Le système de guidage inertiel qui se trouvait dans la fusée Ariane 5 est le même que celui qui équipait les précédents modèles de la fusée Ariane. Le plan de vol d'Ariane 5 diffère de celui d'Ariane 4, sa trajectoire est différente et les accélérations sont 5 fois plus fortes. Les valeurs élevées mesurées par les accéléromètres ont provoqué un dépassement de capacité lors du calcul de la position géographique de la fusée par le dispositif informatique du système de guidage. Tout comme pour Ariane 4, le système de guidage inertiel est en mode de calibrage durant les premières secondes du vol, quand l'incident s'est produit. Le calibrage est maintenu jusqu'au démarrage du pilote automatique après 40 secondes de vol. Sur Ariane 5, il n'est plus nécessaire de maintenir le mode de calibrage au début du vol, il a néanmoins été maintenu pour des raisons de commodité. L'ordinateur de bord reconnait la défaillance du système de guidage inertiel principal. En cas de défaillance, l'ordinateur de bord se branche au système de guidage inertiel de secours. Il n'a pas reconnu la défaillance du système de secours, et interprété le signal de panne de celui-ci comme une information de trajectoire de la fusée – trajectoire très différente de celle que la fusée suivait. L'ordinateur a alors commandé une violente correction de trajectoire suite à une déviation qui n'a en fait jamais eu lieu. Le virage serré demandé par l'ordinateur de bord a provoqué un dérapage de la fusée. L'angle d'incidence de plus de 20 degrés dû au dérapage a arraché un des moteurs d'appoint. Un interrupteur de sécurité déclenche l'auto-destruction préventive de la fusée en cas de détachement accidentel des moteurs d'appoint à basse altitude, ceci en vue d'éviter des victimes au sol suite à un crash de la fusée.

I.2.3 Enquête

Le vol a été largement suivi, par caméra, radar, et télémesures et le dysfonctionnement du système de guidage inertiel a été rapidement cerné par l'équipe d'enquête comme étant la cause de l'incident. Les informations des télémesures ont été envoyées au Centre national d'études spatiales de Toulouse, en France pour analyse, tandis qu'une équipe sur place s'affairait à récupérer les débris de la fusée. La priorité a été donnée aux débris qui présentaient un risque d'incendie, tels que des réserves de carburant non brûlé. La récupération des débris a été rendue difficile, du fait que cette région est essentiellement composée de mangroves et de savanes, gorgées d'eau après la saison des pluies qui venait de se terminer. Des pièces lourdes telles que les tuyères – pesant plusieurs tonnes – ont été retrouvées sous plusieurs mètres d'eau, profondément enfoncées dans la vase, et n'ont jamais été retirées. La récupération des deux systèmes de guidage inertiel parmi les débris de la fusée, et l'analyse des informations encore présentes dans la mémoire des appareils a permis de retracer avec précision les dernières secondes du vol. L'enquête s'est portée sur le cahier des charges du système de navigation, et les essais en laboratoire nécessaires pour obtenir l'autorisation de vol. Des simulations de vol après-coup, utilisant les systèmes de guidage inertiel et l'ordinateur de bord, dans les conditions de vol d'Ariane 5 ont reproduit les événements qui ont conduit à l'explosion de la fusée. Les résultats correspondaient aux informations retrouvées dans les mémoires des appareils qui ont servi durant le vol.

I.2.4 Conclusions

Dans le rapport de la commission d'enquête, les points suivants ont été soulevés :

- Des simulations en laboratoire ont en principe lieu avant le décollage. La réussite des simulations est nécessaire pour obtenir le certificat de vol. Le système de navigation, utilisé depuis longtemps sur Ariane 4, étant réputé fiable, le Centre national d'études spatiales a demandé de ne pas faire de simulations de vol pour ces appareils, et ainsi économiser 800 000 frs sur le coût des préparatifs.
- Réalisées après la catastrophe, les simulations en laboratoire ont permis de vérifier que l'explosion était inéluctable.
- Le bug informatique qui a causé la mise hors service des systèmes de guidage par inertie a eu lieu durant la procédure d'étalonnage de l'appareil. L'étalonnage mesure de très faibles valeurs lorsque la fusée est immobile, et n'est pas protégée contre les valeurs élevées qui peuvent être mesurées durant le vol.
- Le choix de laisser l'appareil en mode calibrage après le décollage date du début du programme Ariane, plus de dix ans avant l'incident. Il est motivé par le fait que sur les premières fusées Ariane, en cas de retardement du décollage, il était nécessaire de relancer le calibrage, qui durait plus de 45 minutes. Ce n'est plus le cas avec Ariane 5.
- L'arrêt automatique du système de guidage inertiel principal en cas d'avarie était un choix de conception décidé longtemps avant l'incident. L'avarie simultanée des deux systèmes de guidage (principal et secours) n'avait pas été envisagée et les conséquences n'avaient pas été anticipées avant le décollage de Ariane 5.
- D'une manière générale, le cahier des charges du programme Ariane est tourné sur les erreurs aléatoires et momentanées des appareils. Dans ces cas, un appareil de secours identique fait généralement l'affaire. En cas de défaut de construction des appareils, deux appareils identiques sont susceptibles de tomber en panne en même temps.
- Une centrale inertielle mesure des grandeurs physiques telles qu'accélération et vitesse angulaire, qui sont très difficiles à reproduire en laboratoire. Les essais en laboratoires consistent à remplacer les mesures de la centrale inertielle par de fausses valeurs. La vérification de l'ensemble centrale inertielle + ordinateur de bord ne peut se faire que lors des vols d'essai (une pratique courante dans l'industrie aéronautique et astronautique).
- Selon le cahier des charges de la centrale inertielle, une erreur informatique au sein de l'appareil doit entraîner son arrêt immédiat, l'erreur doit être inscrite dans une mémoire permanente de l'appareil (EEPROM), et un signal de panne doit être transmis aux autres appareils. C'est le choix de conception d'arrêter l'appareil en cas d'incident qui a été fatal à Ariane 5.
- Le calculateur de la centrale inertielle est équipé de protections qui évitent qu'une erreur informatique se produise en cas de mesure trop élevée. Pour certaines valeurs, il est physiquement impossible d'atteindre la limite, ou alors il existe une large marge de sécurité, et il a été décidé de ne pas mettre de protection. Ces décisions de protection ont été prises de concert entre différents contractants, durant le programme spatial Ariane 4. C'est le dépassement d'une valeur non protégée qui a provoqué l'incident.
- Après enquête, les ingénieurs du CNES se sont aperçus que par mesure d'économie,

le logiciel de navigation de la fusée Ariane 5 était celui qui avait été conçu pour Ariane 4. Mais cela a suffi pour créer une incompatibilité entre le logiciel et le matériel.

`Au moins essayez de lire ce passage...`

Tout tenait à une seule petite variable : celle allouée à l'accélération horizontale. En effet, l'accélération maximum d'Ariane 4 était d'environ 64, la variable a été codée sur 8 bits. Dans un ordinateur, les informations sont codées dans un alphabet un peu spécial appelé langage binaire. Un bit équivaut à une lettre d'un alphabet contenant les deux lettres "0" et "1" ; ainsi tout mot (ou valeur de variable) s'écrit par combinaison de ces deux lettres. Donc un mot de 8 bits s'écrit par une combinaison de 8 lettres, chacune de ces lettres étant soit un "0" soit un "1". En base binaire, cela fait $2^8 = 256$ valeurs possibles (256 combinaisons de 8 bits), suffisant pour coder la valeur 64 qui s'écrit 1000000 et nécessite 8 bits. Mais Ariane 5 était plus véloce : son accélération pouvait atteindre la valeur 300 (qui vaut 100101100 en binaire et nécessite 9 bits). Ainsi, la variable codée sur 8 bits a connu un dépassement de capacité puisque son emplacement mémoire n'était pas assez grand pour accepter une valeur aussi grande, il aurait fallu la coder sur un bit de plus, à savoir 9, ce qui nous aurait fait $2^9 = 512$ comme valeur limite, alors suffisant pour coder la valeur 300. De ce dépassement de capacité a résulté une valeur absurde dans la variable, ne correspondant pas à la réalité. Par effet domino, le logiciel décida de l'autodestruction de la fusée à partir de cette donnée erronée.

I.3 Un lien qui fonctionne

Le rapport d'enquête dans son ensemble

I.4 Comment se faciliter la réalisation de ce projet

`Si vous venez d'arriver sur cette partie sans lire ce qu'il y a avant, vous êtes niais.`

Ce projet est plus facile à réaliser si vous vous dites une bonne fois pour toute que si vous faite 3 tests à la con au fur et à mesure, vous éviterez le crash !

Chapitre II

Introduction

L'existence des shells est liée à l'existence même de l'informatique. L'idée que `communiquer avec un ordinateur en utilisant des interrupteurs 0/1 alignés est chiant` s'est rapidement imposée parmis les developpeurs de l'époque. Du coup, l'idée que `communiquer avec un ordinateur en utilisant une ligne de commandes interactive proche de l'anglais serait vachement plus pratique` s'est imposée très naturellement.

Avec `Minishell`, vous allez pouvoir faire un voyage dans le temps et revenir à des problématiques de l'époque ou `Windows` n'existait pas. Si ça ne fait pas de vous un meilleur développeur, alors rien ne le fera.

Chapitre III

Objectifs

Le projet `Minishell` va vous permettre de vous plonger au coeur d'un système `Unix` et de découvrir une partie importante de l'API d'un tel système : la création et la synchronisation de processus. Le lancement d'une commande dans un shell implique la création d'un nouveau processus dont le processus parent doit monitorer l'exécution et l'état final. Cet ensemble de fonctions sera le coeur de votre `Minishell`, assurez-vous donc de coder le plus proprement et de la manière la plus organisée possible car sinon, vous devrez tout refaire pour le projet `21sh`. Ca serait dommage non ?

Soyez donc rigoureux dans votre pratique du `C`, prenez le temps nécéssaire à la lecture et la compréhension des différents `mans`, et surtout, surtout, testez votre code !

Chapitre IV

Consignes

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- L'exécutable doit s'appeller `minishell`.
- Vous devez rendre un `Makefile`. Ce `Makefile` doit compiler le projet, et doit contenir les règles habituelles. Il ne doit recompiler et relinker le programme qu'en cas de nécessité.
- Si vous êtes malin et que vous utilisez votre biliothèque `libft` pour votre `minishell`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépôt de rendu. Votre `Makefile` devra compiler la bibliothèque, en appelant son `Makefile`, puis compiler votre projet.
- Votre projet doit être en `C` et à la Norme. La norminette fait foi.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (segmentation fault, bus error, double free, etc...).
- Votre programme ne doit pas avoir de fuites mémoire.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes :
  - `malloc`,`free`
  - `access`
  - `open`, `close`, `read`, `write`
  - `opendir`, `readdir`, `closedir`
  - `getcwd`, `chdir`
  - `stat`, `lstat`, `fstat`
  - `fork`, `execve`

  - `wait`, `waitpid`, `wait3`, `wait4`
  - `signal`, `kill`
  - `exit`

- Vous avez l'autorisation d'utiliser d'autres fonctions dans le cadre de vos bonus, à condition que leur utilisation soit dûment justifiée lors de votre correction. Par exemple, utiliser `tcgetattr` est justifiable dans certains cas, utiliser `printf` par flemme ne l'est jamais. Soyez malins.

- Vous pouvez poser vos questions sur le forum, sur slack, ...

Chapitre V

Sujet - Partie obligatoire

- Vous devez programmer un mini interpréteur de commandes `UNIX`.
- Cet interpréteur doit afficher un prompt (un simple `"$> "` par exemple) et attendre que vous tapiez une ligne de commande, validée par un retour à la ligne.
- Le prompt n'est de nouveau affiché qu'après la fin de l'exécution de la commande.
- Les lignes de commande sont simples, pas de pipes, pas de redirections ou autres fonctions avancées.
- Les exécutables sont ceux que l'on peut trouver dans les chemins indiqués dans la variable `PATH`.
- Dans le cas où l'exécutable ne peut être trouvé, il faut afficher un message d'erreur et réafficher le prompt.
- Vous devez gérer les erreurs sans utiliser `errno`, en affichant un message adapté sur la sortie d'erreur.
- Vous devez gérer correctement le `PATH` et l'environnement (copie du `char **environ` système).
- Vous devez implémenter une série de builtins : `echo`, `cd`, `setenv`, `unsetenv`, `env`, `exit`.
- Vous choisissez le shell de référence que vous souhaitez.

`Lisez bien les man.`

Voici un exemple d'utilisation de votre `minishell` :

```
$> cd /dev
$> pwd
/dev
$> ls -l
total 0
crw-rw----  1 root   video     10, 175 dec 19 09:50 agpgart
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrom -> hdc
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrom0 -> hdc
drwxr-xr-x  2 root   root           60 dec 19 09:50 cdroms/
lrwxrwxrwx  1 root   root            3 dec 19 09:50 cdrw -> hdc
lrwxrwxrwx  1 root   root           11 dec 19 09:50 core -> /proc/kcore
drwxr-xr-x  3 root   root           60 dec 19 09:50 cpu/
drwxr-xr-x  3 root   root           60 dec 19 09:50 discs/
lrwxrwxrwx  1 root   root            3 dec 19 09:50 disk -> hda
lrwxrwxrwx  1 root   root            3 dec 19 09:50 dvd -> hdc
lrwxrwxrwx  1 root   root            3 dec 19 09:50 dvdrw -> hdc
crw-------  1 root   root      29,   0 dec 19 09:50 fb0
lrwxrwxrwx  1 root   root           13 dec 19 09:50 fd -> /proc/self/fd/
brw-rw----  1 root   floppy     2,   0 dec 19 09:50 fd0
brw-rw----  1 root   floppy     2,   1 dec 19 09:50 fd1
crw-rw-rw-  1 root   root       1,   7 dec 19 09:50 full
brw-rw----  1 root   root       3,   0 dec 19 09:50 hda
brw-rw----  1 root   root       3,   1 dec 19 09:50 hda1
brw-rw----  1 root   root       3,   2 dec 19 09:50 hda2
brw-rw----  1 root   root       3,   3 dec 19 09:50 hda3
brw-rw----  1 root   root       3,   5 dec 19 09:50 hda5
brw-rw----  1 root   root       3,   6 dec 19 09:50 hda6
$> kwame
kwame: command not found
$>
```

Chapitre VI

Sujet - Partie bonus

Pas mal de features seront au menu des 21sh et 42sh. Voici néanmoins une liste de bonus que vous pouvez implémenter dès maintenant si le coeur vous en dit :

- La gestion des signaux, et notamment `Ctrl-C`. L'utilisation de variables globales est autorisé dans le cadre de la gestion des signaux.
- La gestion des droits d'exécution dans le PATH.
- La complétion.
- La séparation des commandes avec le " ;".
- D'autres bonus que vous jugez utiles.

Chapitre VII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépôt `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué.

Bon courage à tous et n'oubliez pas votre fichier auteur !