

Nanoshell

`fork()` + `execve()` + `waitpid()` is all you need.

Introduction

L'objectif de ce projet est d'implémenter un **shell UNIX**.



Il y a environ 10.000 features possibles à implémenter dans un shell. Le vôtre n'en retiendra que les fonctionnalités essentielles : le but est de comprendre **comment les fonctions `fork()`, `execve()` et `waitpid()` s'emboîtent !**



Le projet

Vous devez écrire un programme qui puisse être raisonnablement appelé un "shell". Pour mériter cette appellation, voici les fonctionnalités à gérer :

- Quand vous lancez votre programme, vous **affichez un prompt**.
- L'utilisateur **saisit une chaîne de caractères** et appuie sur "Entrée" (on ne vous demande pas une gestion avancée de la ligne de saisie : ni déplacement, ni raccourcis, etc... [Curieux ?](#))
- Si la chaîne décrit un **exécutable présent dans le répertoire courant ou dans le path**, celui-ci est exécuté, avec les paramètres saisis
- Si la chaîne est un **builtin** reconnu, ce builtin est exécuté avec les paramètres. Builtins à implémenter : cd, exit, pwd, env
- Sinon, votre shell doit afficher une erreur :

```
?> grutcblu  
nanoshell: weird, grutcblu is not here... :/
```

Concernant cd, on vous demande simplement de gérer "cd ..", "cd ~", "cd" tout court, et cd somedir (un répertoire du dossier courant, pas de chemin sophistiqué type ".../path/...").

Implémentation

Lecture de la ligne :

- Vous lisez sur le file descriptor stdin (càd 0)



- Vous pouvez découper la ligne par espaces (on ne demande pas de parsing sophistiqué)
- Pas de gestion des guillemets requise, ni d'échappement de caractères
- Une ligne vide ne fait rien

Gestion du PATH :

Si la commande contient un “/”, tenter d'exécuter la commande dans le répertoire courant. Sinon :

- Lire la variable d'environnement PATH
- Tester chaque répertoire qui s'y trouve
- Construire le chemin complet du binaire avec la première “solution” trouvée (ex. : /usr/local/bin/python3)

Fonctions autorisées :

- `printf()` et `fgets()` (impression texte et saisie de l'input utilisateur)
- `malloc()` et `free()`
- `fork()` et `execve()`
- `waitpid()`
- `exit()`
- `getenv()` (parcours de l'environnement pour récupérer le PATH)
- `perror()`
- `getcwd()` (obtention du répertoire courant du process)
- `chdir()` (changement du répertoire courant)
- `access()` (test si un chemin est exécutable)
- `str*` () (toutes les fonctions sur les strings)

Par défaut, une fonction qui n'est pas dans cette liste n'est pas autorisée.



Compilation et organisation du code

- Votre code doit se trouver dans un répertoire `src/`
- Votre projet doit utiliser un `Makefile`
- Ce `Makefile` doit compiler tout votre code et produire un exécutable qui s'appelle `nanoshell` (ou `nanoshell.exe` sur Windows)
- Votre code doit compiler avec les flags `-Wall -Wextra -Werror` (on conseille de désactiver `-Werror` pendant le développement, et ne l'activer qu'à la fin !)
- Vous devez utiliser un fichier de header `nanoshell.h`, que vous importerez depuis vos fichiers de code. Il contiendra tous les prototypes de vos fonctions, et devra se prémunir contre l'inclusion multiple

Aller plus loin

- Ctrl-D (EOF) quitte le shell proprement
- Épurage et parsing avant d'exécuter la commande
- Gestion des esperluettes
- [Gestion des pipes](#)
- Gestion des redirections

Compétences visées

- C
- environnement UNIX



Rendu

Le projet est à rendre sur votre github :

<https://github.com/prenom-nom/nanoshell>

Base de connaissances

- <https://man7.org/linux/man-pages/man2/execve.2.html>
- <https://man7.org/linux/man-pages/man2/fork.2.html>
- <https://man7.org/linux/man-pages/man2/wait.2.html>
- <https://man7.org/linux/man-pages/man2/getpid.2.html>
- <https://man7.org/linux/man-pages/man2/chdir.2.html>
- https://en.wikipedia.org/wiki/Shell_builtin