Projet : minishell

1 Objectifs

Ce projet vise à vous permettre de mettre en pratique les notions de base vues en TD et TP, autour de la gestion des processus, des signaux et des E/S, dans un contexte suffisamment réaliste. Il s'agit plus précisément de développer un interpréteur de commandes simplifié, offrant les fonctionnalités de base des shells Unix, comme le bash...

Les différents parties du projet seront à développer en séance de TP.

2 Boucle principale

La première étape va être d'écrire le comportement de base de l'interpréteur, qui consiste en une boucle infinie. Chaque itération de cette boucle lit une ligne sur l'entrée standard, l'interprète comme une commande, puis lance un processus fils qui exécutera cette commande.

Par exemple, lors de la session suivante :

```
sh-3.2$ pwd
/Users/jeromeermont/Cours/Système 1SN/minishell
sh-3.2$ ls
Makefile  minishell.c readcmd.c readcmd.h test_readcmd.c
sh-3.2$ wc -l minishell.c
      52 minishell.c
sh-3.2$
```

L'interpréteur lit, reconnaît puis lance l'exécution des commandes `pwd`, `ls`, `wc`, puis reste en attente de lecture de la ligne suivante.

2.1 Lancer une commande élémentaire

Question 1 (Lancement d'une commande) Réaliser la boucle de base de l'interpréteur, en se limitant à des commandes simples (pas d'opérateurs de composition), sans motifs pour les noms de fichiers.

Le projet étant centré sur les fonctions et notions système, et non sur l'analyse lexicale,

- on ne considèrera pas par la suite la possibilité de définir des motifs pour les noms de fichiers ;
- une fonction de lecture et d'analyse d'une ligne saisie sur l'entrée standard est fournie (fichiers `readcmd.c` et `readcmd.h`). Il n'est pas nécessaire [1] pour la suite de modifier le code de `readcmd.c`.

2.2 Synchronisation entre le shell et ses fils

Si, comme cela est demandé pour la question 1 le processus shell lance un fils, puis se met immédiatement en attente de lecture de la prochaine ligne de commande, il est possible que l'affichage de l'invite précède ou se mêle à l'exécution du processus fils.

Question 2 (Exemple) Construire une session simple (utilisant le code écrit pour la question 1) mettant en évidence ce comportement.

Question 3 (Enchaînement séquentiel des commandes) Modifier votre code afin qu'il attende la fin de la dernière commande lancée avant de passer à la lecture de la ligne suivante.

1. et même carrément déconseillé, si l'on ne souhaite pas s'engager dans le développement d'un analyseur syntaxique...

Question 4 (Lancement de commandes en tâche de fond) Le comportement du code initial (celui écrit en réponse à la question 1) correspond cependant à une possibilité utile offerte par les shells, à savoir le lancement de commandes en tâche de fond, spécifié par un & en fin de ligne. Compléter votre code pour offrir cette possibilité.
Note : Le parseur fourni (fichiers `readcmd`) permet d'analyser et reconnaître un & en fin de ligne.

3 Gestion des processus lancés depuis le shell

Il s'agit ici de réaliser une version simplifiée des commandes du shell qui facilitent la gestion des processus lancés depuis l'interpréteur de commandes lui-même.

Question 5 (Gérer les processus lancés depuis le shell) Compléter votre code par les commandes internes suivantes :

- `lj`, (*list jobs*) qui donne la liste des processus lancés depuis le minishell (« *jobs* ») et non encore terminés, avec leur identifiant propre au minishell, leur pid, leur état (actif/suspendu) et la ligne de commande lancée.
- `sj`, (*stop job*)qui permet de suspendre un job (l'identifiant à fournir à la commande `sj` sera l'identifiant géré par le minishell).
- `bg`,(*background*) qui permet de reprendre en arrière-plan (en tâche de fond) un job suspendu (l'identifiant à fournir à la commande `bg` sera l'identifiant géré par le minishell).
- `fg`, (*foreground*) qui permet de poursuivre en avant-plan un job suspendu ou en arrière-plan (l'identifiant à fournir à la commande `fg` sera l'identifiant géré par le minishell).

Indication : la mise en œuvre des commandes `lj, sj, fg, bg` pourra s'appuyer sur l'utilisation de la primitive `waitpid(_)`, et l'exploitation du signal `SIGCHLD`. Le TP signaux aborde ce point technique, qui est développé et détaillé dans une note disponible sur Moodle , dans la section consacrée au projet.

4 Contrôle des processus en avant-plan via le terminal

La frappe du caractère `ctrl-Z` [resp. `ctrl-C`] provoque l'envoi du signal `SIGTSTP` [resp `SIGINT`] aux processus en avant plan. On souhaite retrouver ce comportement au niveau du minishell.

Question 6 (`SIGTSTP`) Comme pour le shell standard, un processus lancé depuis le minishell pourra être suspendu par

- la commande `sj` ;
- l'envoi du signal `SIGSTOP` ;
- **ou** *la frappe de `ctrl-Z` au clavier s'il s'agit d'un processus en avant-plan du minishell.*

La frappe de `ctrl-Z` ne doit pas provoquer la suspension de votre shell, ni celle de ses processus en arrière-plan, mais devra amener la suspension du processus en avant-plan (éventuel) de votre shell. Compléter votre programme pour traiter cette frappe en conséquence.

Afin de faciliter la mise en œuvre, vous ***pourrez*** en revanche faire, à votre convenance (mais en justifiant vos choix), l'hypothèse que pour les processus lancés depuis le minishell, le signal `SIGTSTP` est ignoré, et/ou qu'ils ne démasquent pas ce signal, et/ou ne redéfinissent pas le traitant de ce signal.

Complétez ce code par l'ajout d'une commande interne `susp` qui suspend le minishell (et lui seul).

Question 7 (`SIGINT`) la frappe de `ctrl-C` au clavier se traduit par l'envoi à votre shell du signal `SIGINT`. La réception de ce signal ne doit pas provoquer la terminaison de votre shell, ni celle de ses processus en arrière-plan, mais devra amener la terminaison du processus en avant-plan (éventuel) de votre shell. Compléter votre programme pour traiter cette frappe en conséquence.

Comme pour la question précédente, afin de faciliter la mise en œuvre, vous ***pourrez*** faire, à votre convenance (mais en justifiant vos choix), l'hypothèse que pour les processus lancés depuis le minishell, le signal `SIGINT` est ignoré, et/ou qu'ils ne démasquent pas ce signal, et/ou ne redéfinissent pas le traitant de ce signal.

5 Gestion des redirections

Question 8 (Redirections) Compléter votre programme pour permettre d'associer l'entrée standard ou la sortie standard d'une commande à un fichier.
Exemple (classique) : `cat < f1 > f2` associe `f1` à l'entrée standard de `cat`, et `f2` à la sortie standard de `cat`.

6 Tubes

Question 9 (Tubes simples) Compléter votre programme pour permettre de composer des commandes en les reliant par un tube.

Exemple : `ls | wc -l` lance les commandes `ls` et `wc`, la sortie standard de `ls` étant connectée à l'entrée standard de `wc` par un tube.

Question 10 (Pipelines) Etendre la fonctionnalité précédente en offrant la possibilité d'enchaîner une séquence de filtres liés par des tubes, de sorte à obtenir un traitement en pipeline.

Exemple : `cat toto.c lulu.c | grep int | wc -l`

7 Guide de progression

Les différentes questions seront traitées en TP :
Questions 1,2,3,4 : TP1 processus ;
Questions 5,6 : TP2 signaux ;
Questions 7 : TP3 signaux ;
Question 8 : TP4 fichiers, TP5 redirections ;
Questions 9,10 : TP6 tubes ;

8 Modalités pratiques

Attention : Les modalités pratiques peuvent évoluer en fonction des circonstances et du déroulement de l'enseignement. Les **modalités à jour** sont disponibles sur la **page Moodle** (section projet et pages des liens de dépôt).

Pour chaque étape du projet, développée en séance de TP, vous rendrez une archive contenant les fichiers. Ces différentes étapes seront évaluées et seront intégrées à la note finale du projet.

Le ... vous devrez rendre votre une version du minishell correspondant au traitement complet. Les livrables pour ce rendu seront constitués du code et d'un bref rapport présentant l'architecture de l'application, les choix et spécificités de conception, la **méthodologie de tests** suivie, avec quelques test significatifs. Le rapport devra en particulier comporter la réponses à la question 2, ainsi qu'une explication sur les choix opérés pour la mise en œuvre des réponses aux questions 7 et 8. L'évaluation du code tiendra compte de sa qualité.

Attention : ce projet est un projet individuel. Il est permis, et même bénéfique, d'échanger sur la conception, mais le rendu final doit refléter un travail strictement personnel. Nous vous demandons de bien noter que nous disposons d'outils spécifiques pour détecter la copie de projets, et que **toute fraude avérée sera sanctionnée sans appel, quelles que soient les circonstances.**