MiniShell : travail à rendre

La ressource R3.05 de programmation système sera, pour partie, évaluée par le travail réalisé sur le projet `MiniShell` développé en `C` au cours du module (l'autre partie de l'évaluation étant une interrogation portant sur les principaux concepts de la programmation système). **On vous demande de travailler en équipe (2-3 étudiants).**

Les critères d'évaluation sont :

- l'aboutissement des différents objectifs,
- la progression dans les développements,
- la qualité du code des extensions,
- les tests et la gestion de votre projet sous `Git`.

A partir des spécifications proposées au cours du module R3.05 (en particulier, la refactorisation du code à l'aide d'`Actions`), on vous demande de poursuivre le développement du `Shell` en utilisant les concepts de programmation système. Voici une liste de contributions attendues (mais vous pouvez ajouter des éléments à cette liste) :

- **pouvoir lancer des commandes internes** (par exemple : `cd`, `pwd`, `mkdir` ...) [**requis**],
- **pouvoir lancer des commandes externes** (par exemple : `ls`, `cp`, `rm` ...), elles seront préfixées par le marqueur '!' [**requis**],
- **gérer les travaux** avec la commande `jobs` lorsque les commandes externes sont lancées de manière asynchrone (en mode détaché avec le suffix '&'), attention à la terminaison avec le signal `SIGCHLD`, vous utiliserez une structure de données adaptée pour stocker les informations sur les processus en cours d'exécution [**requis**],
- **gérer les redirections des flux d'entrée/sortie** depuis/vers des fichiers avec les marqueurs '<' et '>' [**optionnel**],
- **gérer la communication entre processus** à l'aide d'un tube (`pipe`), les commandes seront alors connectées avec le marqueur '|', la sortie standard de la commande précédent le `pipe` sera connectée à l'entrée standard de la commande suivante [**optionnel**].

Pour aller plus loin, vous pouvez faire évoluer le `MiniShell` afin de répondre aux fonctionnalités suivantes :

- gérer un fichier de configuration (par exemple ~`/.minishellrc`) [**optionnel**] :
  - avec des **variables d'environnement**, par exemple :

```
export EDITOR=emacs
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/cuda/lib64
```

  - avec des **alias**, par exemple :

```
alias cat='bat --style=plain'
alias ll='exa -al -snew'
```

- la méthode `split_line` mise en place pour analyser une commande saisie par l'utilisateur est très basique, elle se contente de décomposer des mots séparés par un espace. On vous propose de reprendre cette partie du code pour construire un véritable **arbre syntaxique** en s'appuyant sur les marqueurs standards d'un `Shell` (opérateurs binaires '<','>','|',... ou unaires '!','&',...), vous utiliserez une structure de données adaptée (voir `https://docs.gtk.org/glib/#structs`) [**optionnel**].

Ce travail est à rendre, au plus tard, le **lundi 7 novembre 2022**, dans un projet hébergé sur le `Gitlab` de l'IUT. Pensez à inviter (avec un rôle ≥ `Reporter`) votre enseignant responsable du module R3.05.