Système d'Exploitation - L2, S4
TP 1 - Processus 1 : Invite de Commandes

Un interpréteur de commandes (ou `shell`) est un programme permettant d'exécuter des commandes de manière interactive. L'invite (ou *prompt*) est le message qui s'affiche lorsque l'interpréteur de commandes est prêt à exécuter une commande. L'objectif de ce TP est d'implémenter un interpréteur de commandes.

Remarque.

— Consultez la documentation des fonctions que vous ne connaissez pas. Si vous voulez avoir la documentation sur une fonction système, *e.g.*, `read`, c'est `man 2 read`, et les fonctions de la librairie standard c'est dans la section 3, *e.g.*, pour `strdup`, `man 3 strdup`. N'oubliez pas que `man 2 intro` donne une documentation sur la gestion des erreurs par les fonctions systèmes.

— Les commandes et fonctionnalités implémentées sont similaires à celles implémentées dans les interpréteurs habituels. N'hésitez pas à tester les fonctionnalités sur votre interpréteur habituel avant de les implémenter dans le vôtre. Si vous voulez avoir la documentation sur une commande, *e.g.*, `ls`, c'est `man ls`.

Exercice 1.1 *Affichage invite et saisie de commandes*

1. Écrivez un programme qui affiche une invite et qui lit une chaîne de caractères (contenant éventuellement des espaces). Votre programme ne doit pas être sujet à des failles de type *buffer overflow*, lorsque la chaîne de caractères saisie dépasse la taille maximale allouée.
2. Modifiez votre programme pour qu'il affiche l'invite jusqu'à ce que l'utilisateur tape `quit` ou `exit`.

Exercice 1.2 *Exécution de commandes simples*

La partie suivante d'un programme exécute la commande `ls -a -l`.

```
char * arguments[4];
arguments[0] = strdup("ls");
arguments[1] = strdup("-a");
arguments[2] = strdup("-l");
arguments[3] = NULL;
execvp(arguments[0], arguments);
```

1. Modifiez votre programme de manière à exécuter une commande appelée par l'utilisateur. Assurez-vous que votre programme gère correctement un nombre d'espaces quelconque. Par exemple, lorsque l'utilisateur entre `ls  -l  `, la commande à exécuter est `ls -l`, *i.e.*, il faut supprimer les espaces inutiles.
2. L'exécution d'une commande par `execvp` écrase le code du programme l'appelant. Pour pouvoir exécuter une commande sans perdre le code du programme l'appelant, il est

nécessaire d'utiliser deux processus : le fils exécutant la commande (son code étant écrasé) et le père attendant que le fils se termine (son code étant conservé). La partie suivante d'un programme lance deux processus, le fils attendant cinq secondes et le père attendant que le fils termine.

```
int pid = fork();
if (pid<0) {
printf("error: fork");
}
else if (pid==0) {
printf("child - begin\n");
sleep(5);
printf("child - end\n");
}
else {
waitpid(pid, NULL, 0);
printf("father - continuing\n");
}
```

Modifiez votre programme de manière à exécuter des commandes appelées par l'utilisateur.

Exercice 1.3 *Exécution de commandes en arrière plan*

1. Modifiez votre programme de manière à ce que l'exécution d'une commande terminant avec un `'&'` rende la main immédiatement à l'interpréteur de commandes, en laissant le programme appelé s'exécuter en arrière plan.
2. Testez votre programme avec `xeyes &`.

Exercice 1.4 *Gestion des répertoires*
La commande `which` indique dans quel répertoire se trouve un programme.

1. Dans quel répertoire se trouve le programme `ls` ?
2. Dans quel répertoire se trouve le programme `pwd` ?
3. Est-ce que `cd` est un programme ?
4. Est-ce que votre interpréteur comprend la commande `cd` ?
5. Modifiez votre programme de manière à ce qu'il gère la commande `cd`. Vous pourrez utiliser la fonction système `chdir`.

Exercice 1.5 *Gestion des processus zombies*

1. Exécutez `xeyes &`, et fermez ce programme. Exécutez la commande `ps a` et observez l'état du processus `xeyes`. Est-ce que vous pouvez observer que le processus `xeyes` est devenu un processus zombie ?
2. Pourquoi le fait de quitter votre interpréteur fait que les processus zombies disparaissent ?

3. Modifiez votre programme de manière à ce que tous les processus zombies soient supprimés à chaque fois qu'une commande est tapée (avant l'exécution de la commande, ou après). Pour cela, relisez la documentation sur `waitpid`.
4. Vérifiez que votre programme est correct en exécutant deux fois `xeyes &`, puis en les quittant tous les deux et en tapant une seule commande. Les deux processus zombies doivent avoir disparu.
5. Pouvait-on éviter les processus zombies ? Si oui, comment ?

Exercice 1.6 *Optionnel*

1. Les processus communiquent à travers des tubes. Implémenter le mécanisme d'enchaînement de processus au travers de tubes de communication. Pour ce faire, vous pouvez par exemple pour chaque commande (sauf la dernière), installer un tube entre cette dernière et la commande suivante, lancer un nouveau processus pour la commande courante et passer à la commande suivante.
2. Lancez votre interprète shell, exécutez une commande (par exemple cat) et tapez par exemple Ctrl-C ? Que remarquez-vous ? Faites en sorte que les signaux Ctrl-C et Ctrl-Z (stopper un processus) soit envoyé aux processus lancés par votre shell.