TP de systèmes d'exploitation et programmation concurrente
Mini-shell
ENSIMAG 2ème année

François Broquedis, Hugues Evrard, Grégory Mounié, Vivien Quéma *

1 Présentation du TP

Le but de ce TP est de vous familiariser avec la gestion des processus sous UNIX en vous faisant écrire un petit interpréteur de commandes, sous ensemble de ce que vous utilisez tous les jours sur ensibm (bash).

La démarche est progressive : le *shell* sera enrichi petit à petit au fur et à mesure de nouvelles fonctionnalités. Le socle est commun. Différentes variantes sur les fonctionnalités sont définies (cf. section 5).

2 Lancement et enchaînement de commandes

Pour commencer vous devez simplement lire des commandes sur l'entrée standard et les exécuter. Pour cela vous devez utiliser les appels système `fork` et `exec` (`man 2 fork`, `man 3 execvp`).

Exemple d'utilisation :

```
> /bin/ls /usr
X11R6  etc    include  kerberos  lib64    local  share  tmp
bin    games  java     lib       libexec  sbin   src
> pwd
/perms/denneuli
```

Question 1 (Lancement d'une commande) *Écrire le programme qui lit des commandes sur l'entrée standard et les exécute. Pour vous aider vous disposez de fonction de lecture de l'entrée standard (fichier* readcmd.c*, exemple d'utilisation dans* ensishell.c*).* readcmd.c *utilise la bibliothèque libre* readline *pour vous permettre l'édition des lignes, la complétion avec le* tab *et la sauvegarde de l'historique.*

2.1 Lancement en tâche de fond

Lancer une commande dont le temps d'exécution est assez long (`sleep 20` par exemple). Vous constaterez que le prompt apparaît immédiatement, sans attendre la fin de l'exécution. En effet par défaut le processus père n'attend pas son fils, celui qu'il a créé. Pour cela il faut utiliser la routine `wait` ou `waitpid` [1].

Question 2 (Attente de la terminaison) *Modifier votre programme pour que lorsqu'une commande est lancée il faille attendre la fin de son exécution pour passer à la suivante.*

Question 3 (Tâche de fond) *Ajouter ensuite la possibilité de lancer une commande en tâche de fond en utilisant le caractère &.*

*d'après un sujet de Yves Denneulin

1. Des détails sur l'appel `waitpid` en tapant `man 2 waitpid`

Exemple :

```
> du -s /tmp &
> ls
....
>
278200  /tmp
```

Le parseur simple fournit ne reconnaît le caractère que si il est seul. Il faut un espace entre le & et l'argument le précédant.

2.2 Liste des processus

Question 4 (Lister les processus en tâches de fond) *Ajouter à votre shell une commande interne* `liste_ps` *qui donne la liste des processus lancés en tâche de fond avec leur pid et la commande lancée. Comment pouvez-vous savoir si ils se sont terminés ? (*`man waitpid`*)*

3 Le pipe

Question 5 (pipe) *Ajouter la possibilité de connecter l'entrée d'un processus avec la sortie d'un autre comme dans la commande* `ls / | grep u`.

La question ne concerne que le cas de 2 commandes seulement (1 seul pipe). Les appels systèmes utiles sont `pipe(...), dup(...), dup2(...), close(...)`.

4 Redirection dans les fichiers

Question 6 (Redirection) *Ajouter la possibilité de connecter l'entrée ou la sortie d'un processus avec des fichiers comme dans la commande* `cat < toto > tata` .

Les redirections et le pipe peuvent être appelés ensemble. Les appels systèmes utiles sont `open(...), dup(...), dup2(...), close(...)`. Elles fonctionnent aussi avec les pipe.

5 Les variantes

La suite du sujet est fonction de votre numéro d'utilisateur (UID, obtenu avec la commande `id -u`) modulo sur le nombre de variantes. C'est l'UID minimum qui est utilisé pour un binôme ou un trinôme. Pour connaître le numéro de votre sujet, il suffit d'exécuter les commandes suivantes sur votre serveur de référence (ensibm).

- Pour un monôme lançant lui-même la commande

  ```
  echo "$(id -u) % 12" | bc
  ```

- Pour un binôme ayant pour identifiants login1 et login2

  ```
  echo "define min(x,y) { if (x<y) return x else return y }; \
          min( $(id -u login1), $(id -u login2)) % 12" | bc
  ```

5.1 Les jokers et les variables d'environnements

Les jokers (l'étoile, les crochets, le tilde, les variables d'environnements) sont remplacés dans la ligne de commande par le shell avant l'exécution.

Question 7 (Joker en wordexp) *Ajouter la possibilité d'utiliser les jokers et les variables d'environnements comme dans la commande* `ls  /t* $PWD/toto`.

Les appels systèmes et fonctions utiles sont `wordexp(...), wordfree(...), strnlen(...), strlcpy(...), strlcat(...), malloc(...), free(...)`.

ID	Variantes
0	Jokers et environnement (sec. 5.1) ; Limitation du temps de calcul (sec. 5.6)
1	Jokers étendus (tilde, brace) (sec. 5.2) ; Pipes multiples (sec. 5.5)
2	Terminaison asynchrone (sec. 5.4) ; Limitations du temps de calcul (sec. 5.6)
3	Temps de calcul (sec. 5.3) ; Pipes multiples (sec. 5.5)
4	Jokers et environnement (sec. 5.1) ; Pipes multiples (sec. 5.5)
5	Jokers étendus (tilde, brace) (sec. 5.2) ; Limitation du temps de calcul (sec. 5.6)
6	Terminaison asynchrone (sec. 5.4) ; Pipes multiples (sec. 5.5)
7	Temps de calcul (sec. 5.3) ; Limitation du temps de calcul (sec. 5.6))
8	Jokers et environnement (sec. 5.1) ; Terminaison asynchrone (sec. 5.4) ;
9	Jokers étendus (tilde, brace) (sec. 5.2) ; Temps de calcul (sec. 5.3) ;
10	Jokers et environnement (sec. 5.1) ; Temps de calcul (sec. 5.3) ;
11	Jokers étendus (tilde, brace) (sec. 5.2) ; Terminaison asynchrone (sec. 5.4) ;

TABLE 1 – Les différentes variantes

5.2 Jokers étendus

Les jokers sont remplacés dans la ligne de commande par le shell avant l'exécution.

Question 8 (Joker en glob) *Ajouter la possibilité d'utiliser les jokers comme dans la commande* `ls ~/t{oto,iti} *.c`.

Les appels systèmes et fonctions utiles sont `glob(...)`, `globfree(...)`, `strnlen(...)`, `strlcpy(...)`, `strlcat(...)`, `malloc(...)`, `free(...)`.

5.3 Temps de calcul d'un processus

Lorsqu'un processus termine, si son père n'est pas en attente de sa terminaison, il lui envoie un signal `SIGCHLD`. La réception du signal peut déclencher un traitant qui réalise un certain nombre d'actions.

Question 9 (Signaux) *Le but est d'afficher un message indiquant le temps de calcul d'un processus lancé en tache de fond (avec un* `&`*) à sa terminaison. Le shell devra afficher immédiatement le message.*

Les appels systèmes et fonctions utiles sont `waitpid(...)`, `sigaction(...)`, `gettimeofday(...)`.
La réception d'un signal débloque les appels systèmes bloquants comme `waitpid` ou `read` (cf leurs manuels). Il faudra les reprendre le cas échéant.

5.4 Terminaison asynchrone

Le but est d'afficher la terminaison d'un processus s'exécutant en tâche de fond au moment où celui-ci se termine, sans attendre le prochain prompt de l'interpréteur demandant une commande.

Question 10 (Signaux) *Il faudra traiter le signal "SIGCHILD" reçu par le père (le shell) lorsque le fils termine. La fonction traitant associée au signal est exécutée de manière asynchrone dès que le signal arrive au processus.*

Les appels systèmes utiles sont `signal(...)`.
La réception d'un signal débloque les appels systèmes bloquants comme waitpid ou read (cf leurs manuels). Il faudra les reprendre le cas échéant.

5.5 Pipes multiples

Question 11 (Pipes multiples) *Le but est que les séquences de pipes multiples comme* `ls -R / | egrep "^to" | egrep "\.jpg$" | gzip -c | gzip -cd | less` *fonctionnent.*

Les appels systèmes utiles sont `pipe(...)`, `dup(...)`, `dup2(...)`, `close(...)`.

5.6 Limitation du temps de calcul

Il est possible de fixer une limite maximum au temps de calcul que les processus lancés peuvent utiliser. À la fin d'un délai souple, le processus reçoit un signal SIGXCPU, qui par défaut le termine. Si il intercepte le signal, il le reçoit ensuite chaque seconde jusqu'à la fin du délai dur, où il est détruit par un signal SIGKILL.

Question 12 (ulimit) *Le but est d'implanter la commande interne* `ulimit X`*, où X est le nombre de secondes avant la limite souple. Mettez la limite dure à X+5 secondes plus tard.*

Attention, la limite ne doit s'appliquer qu'aux nouveaux processus lancés et pas au shell.
Les appels systèmes et fonctions utiles sont `setrlimit(...)`, `strcmp(...)`, `atoi(...)`.