Collomb Sullivan
Titouan Rimbault
Groupe 1

Compte-rendu – Projet MiniShell

Le mini Shell produit au cours de ce projet comporte les caractéristiques suivantes.

1) Interprétation des commandes intégrées:

Ce MiniShell peut **exécuter des commandes qu'il intègre**.

Ces dernières sont:
-**exit** et son alias **quit**, deux commandes permettant de **fermer le MiniShell**.

-Les **commandes de gestion de jobs** décrites dans **g partie 8** de ce compte-rendu.

Si l'utilisateur demande **à exécuter une** (**série de**) **commande**(**s**) et **que cette dernière n'est composée que de commandes intégrées**:
->**Le MiniShell ne génère pas de nouveau job**.

->Les **tests** associés à ces commandes (**exécutées de manière simple**, sans pipes ni redirections) sont décrits dans les fichiers **test_exit.txt**, **test_exit_bis.txt** et **test_commande_integree_ne_cree_pas_de_job.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

2) Interprétation de commandes non-intégrées:

Ce MiniShell peut **exécuter des commandes qu'il n'intègre pas**.

Ces commandes sont **celles décrites dans les dossiers pointés par la variable d'environnement PATH**.

Chaque commande exécutée contenant une commande non-intégrée génère un job. Ces derniers sont listés dans une **variable externe** (voir la **partie 8** pour plus de détails).

```
shell> ls
csapp.c                  fonctions_utiles.c  handlersSignaux.h  listeJobs.c  loop_100
csapp.h                  fonctions_utiles.h  listeInt.c         listeJobs.h  loop_100.c
fichier_sans_lecture.txt handlersSignaux.c   listeInt.h         loop         loop.c
shell>
```

->Les **tests** associés à ces commandes (**exécutées de manière simple**, sans pipes ni redirections) sont décrits dans les fichiers **test_commande.txt**, **test_commande_multiple.txt** et **test_man.txt**.

Gestion des erreurs de noms de commandes:

Si l'utilisateur **demande à exécuter une commande inconnue**, **un message d'erreur est affiché**.

```
shell> okds
Erreur lors du lancement de la commande "okds".
exec: No such file or directory
shell>
```

->Les **tests** associés à ces erreurs (**exécutées de manière simple**, sans pipes ni redirections) sont décrits dans le fichier **test_commande_inexistante.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

3) Interprétation de commandes avec redirections d'entrée et/ou de sortie:

Ce MiniShell peut **exécuter des commandes avec redirection d'entrée et/ou de sortie**. Les **redirections** sont **possibles pour les commandes intégrées** et **non-intégrées**.

```
shell> ls > resultat
shell> cat resultat
csapp.c
csapp.h
fonctions_utiles.c
fonctions_utiles.h
handlersSignaux.c
handlersSignaux.h
listeInt.c
listeInt.h
listeJobs.c
listeJobs.h
loop
loop_100
loop_100.c
loop.c
Makefile
```

->Les **tests** associés à ces commandes (exécutées de manière simples, sans pipes ni redirections) sont décrits dans les fichiers **test_redirection.txt**.

Gestion des erreurs de redirection d'entrée/sortie:

Si l'utilisateur **indique une redirection vers un fichier n'existant pas/sur lequel il n'a pas les droit nécessaires pour effectuer les redirections** (par exemple si l'utilisateur indique une **redirection de la sortie vers un fichier auquel il n'a pas accès en écriture**), **un message d'erreur est affiché**.

```
shell> cat < fichier_inexistant.txt
fichier_inexistant.txt: File not found.
shell>
shell> touch fichier_sans_lecture.txt
shell> chmod 300 fichier_sans_lecture.txt
shell> cat < fichier_sans_lecture.txt
fichier_sans_lecture.txt: Permission denied.
shell>
```

->Les **tests** associés à ces erreurs sont décrits dans les fichier **test_erreur_fichier_entree.txt**, **test_erreur_fichier_sortie.txt** et **test_erreur_fichier_entree02.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

4) Interprétation d'une série de commandes reliées par des tubes:

Ce MiniShell peut **exécuter une série de commandes** dont les différentes parties **sont reliées par des tubes**.

Les **tubes** sont **possibles pour les commandes intégrées** et **non-intégrées**.

```
shell> ls | uniq | grep M | grep M | grep M | grep M | grep M | grep M | grep M | grep M
Makefile
```

```
shell> sleep 50 &
shell> jobs | wc
      1       4      23
```

->Les **tests** associés à cette fonctionnalité sont décrits dans les fichiers **test_pipe_simple.txt**, **test_pipe_simple02.txt** et **test_pipe_multiple.txt**.

5) Exécution de commandes en arrière-plan:

Ce MiniShell peut **exécuter une commande ou une série de commandes en arrière-plan**. C'est à dire que **durant l'exécution de la** (série de) **commande**(s)**, l'utilisateur pourra continuer à utiliser le MiniShell pour effectuer d'autres actions en parallèle**.

```
shell> ps
  PID TTY          TIME CMD
   10 pts/0    00:00:00 bash
 6522 pts/0    00:00:00 shell
 6523 pts/0    00:00:00 ps
shell> ./loop &
shell> ps
  PID TTY          TIME CMD
   10 pts/0    00:00:00 bash
 6522 pts/0    00:00:00 shell
 6524 pts/0    00:00:01 loop
 6525 pts/0    00:00:00 ps
shell>
```

Une **variable externe** a été créee pour **garder en mémoire le numéro du job actuellement en exécution au premier plan**: ***numJobCommandeForeground***.
->Si **aucun job n'est actuellement en exécution au premier plan**, ***numJobCommandeForeground* = -1**.

->Les **tests** associés à cette fonctionnalité sont décrits dans les fichiers **test_execution_background.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

6) Gestion des processus terminés:

Lorsqu'un processus lancé par ce MiniShell se termine, **le MiniShell l'intercepte** afin **d'éviter une invasion de processus zombis**.

Le MiniShell ne rend la main à l'utilisateur suite au lancement d'une commande en foreground que **lorsque tous les processus fils créés pour l'exécution de cette commande** (= lorsque tous les processus associés au jobs de cette commande) **se sont terminés et ont été interceptés**.

```
shell> ls
csapp.c                  fonctions_utiles.c  handlersSignaux.h  listeJobs.c  loop_100
csapp.h                  fonctions_utiles.h  listeInt.c         listeJobs.h  loop_100.c
fichier_sans_lecture.txt  handlersSignaux.c   listeInt.h         loop         loop.c
shell> echo

shell> ls
csapp.c                  fonctions_utiles.c  handlersSignaux.h  listeJobs.c  loop_100
csapp.h                  fonctions_utiles.h  listeInt.c         listeJobs.h  loop_100.c
fichier_sans_lecture.txt  handlersSignaux.c   listeInt.h         loop         loop.c
shell> echo

shell> ps
  PID TTY          TIME CMD
   10 pts/0    00:00:00 bash
 6524 pts/0    00:01:22 loop
 7031 pts/0    00:00:00 shell
 7036 pts/0    00:00:00 ps
```

->Les **tests** associés à cette fonctionnalité sont décrits dans les fichiers **test_gestion_zombie.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

7) Gestion des processus de premier plan:

Lorsqu'un job est **exécuté au premier plan il peut être amené à changer d'état**, suite à la **réception d'un signal *SIGINT*** (**ctrl+C**) ou ***SIGTSTP*** (**ctrl+Z**).

Ces signaux sont donc **traités par le MiniShell**.

Lors de la **réception du signal SIGINT**, le **processus au premier plan** est **terminé et tué**. Lors de la **réception du signal SIGTSTP**, il est **mis en pause** et **placé en arrière-plan**.

```
shell> ./loop
^C
shell> shell> jobs
shell>
```

```
shell> ./loop
^Z
shell> shell> jobs
[1]  Stopped  ./loop
shell>
```

->Les **tests** associés à cette fonctionnalité sont décrits dans les fichiers **test_signal_sigint.txt** et **test_signal_sigtstp.txt**.

Collomb Sullivan
Titouan Rimbault
Groupe 1

8) Gestion des jobs:

Tous les **processus directement lancés par un Shell** sont appelés des **jobs**. Les **jobs associés à notre MiniShell** sont **listés dans une variable externe**: ***listeJobsShell***.

Cette **liste** est accessible à l'utilisateur via **la commande jobs**, qui **affiche sur le terminal la liste des jobs** avec **leur numéro**, **leur état actuel** (en cours d'exécution, arrêté, …) et **la commande ayant créé le job**.

```
shell> ./loop &
shell> sleep 50
^Z
shell> shell> jobs
[1]  Running  ./loop
[2]  Stopped  sleep 50
shell>
```

L'état des jobs peut être modifié via les commandes **fg, bg et stop**:

- **fg** : **exécute au premier plan**, un processus qui **était en arrière-plan**.

```
shell>./loop &
shell> fg 1
```

- **bg** : **relance l'exécution** d'un processus **arrêté** (déjà en **arrière-plan**).

```
shell> ./loop &
shell> stop 1
shell> bg 1
shell> jobs
[1]  Running  ./loop
shell>
```

- **stop** : **met en pause** un processus **en cours d'exécution en arrière-plan**.

```
shell> ./loop &
shell> stop 1
shell> jobs
[1]  Stopped  ./loop
shell>
```

->Les **tests** associés à cette fonctionnalité sont décrits dans les fichiers **test_jobs_fg.txt**, **test_jobs_bg.txt** et **test_jobs_stop.txt**.