

Rapport Final : Jean-Léo DARY

1 SN I

I Architecture de l'application

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.

J'ai implanter une simple boucle `while (stop != 0){}`. Ensuite en lisant attentivement le module `readcmd.[c|h]`. Il était facile d'extraire les commandes des touches tapées par l'utilisateur il suffisait d'écrire les quelques lignes suivantes pour avoir les informations.

```
/* lecture Commande */  
  
commande = readcmd(); /* Lecture de la commande */  
seq = commande->seq;  
backgrounded = commande->backgrounded;
```

Ensuite on exécute les commandes grâce à :

```
error = execvp(seq[0][0], seq[0]);
```

Il fallait que cette ligne de code ne recouvre pas notre programme principal d'où la nécessité d'utiliser des processus fils grâce à la ligne :

```
pid = fork();
```

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

En effet sans :

```
fflush(stdout);
```

les fils récupèrent le tampon non vide ce qui peut entraîner des incohérences au niveau de l'affichage chronologique des informations.

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.

Pour cela il nous faut rajouter un `waitpid` pour attendre que le processus fils finisse la commande. Pour cela j'ai créé une fonction nommée `foreground()` pour désigner les processus d'avant plan (ceux qu'on attend forcément donc) en voici le code :

```
void foreground(int pid) {  
    int status;  
    if (waitpid(pid, &status, 0) < 0)  
    {  
        printf("Erreur waitpid");  
    }  
    if (WIFEXITED(status))  
    {
```

```

        supprimer(pid,&tab_jobs);
    }
    else
    {
    }
}

```

Édit : Certaines lignes ont changé au vu des questions d'après j'ai les ai mises à jour.

Question 4 (Commandes internes) Compléter votre code en ajoutant deux commandes internes, exécutées directement par l'interpréteur sans lancer de processus `_ls` : `cd` et `exit`.

La commande `cd` a eu droit à son propre module comportant 2 fonctions majeures : l'une pour changer le dossier courant l'autre pour le récupérer sous forme de caractère. Le code est commenté voir le module `cd.h` `cd.c` pour plus de détail.

La commande `exit` fut facile à codé puisqu'il faut régler dans ce cas la variable `stop` à 1 pour quitter la boucle.

Question 5 (Lancement de commandes en tâche de fond) Le comportement du code initial (celui écrit en réponse `_a` 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.

Pour mettre en tâche de fond il suffit d'introduire un `if` sur le `waitpid` précédent et donc ne plus attendre la fin du fils. Cette méthode n'est pas sans contrainte puisqu'on a toujours le retour du fils sur l'écran ce qui n'est pas optimal.

Question 6 (Gérer les processus lancés depuis le shell) Compléter votre code par les commandes internes suivantes :

- list**, qui donne la liste des processus lancés depuis le shell et non encore terminées, avec leur identifiant propre au minishell, leur pid, leur état (actif/suspendu) et la ligne de commande lancée.

- stop**, qui permet de suspendre un processus (l'identifiant à fournir à la commande `stop` sera l'identifiant géré par le minishell).

- bg**, qui permet de reprendre en arrière-plan (en tâche de fond) un processus suspendu (l'identifiant à fournir à la commande `bg` sera l'identifiant géré par le minishell).

- fg**, qui permet de poursuivre en avant-plan un processus suspendu ou en arrière-plan (l'identifiant à fournir à la commande `fg` sera l'identifiant géré par le minishell).

La commande `liste` a demandé à faire appel à un module gérant des listes, au vu de sa structure spéciale contenant `id`, `pid`, `cmd`, j'ai préféré l'appeler `processus.[h\c]`. Une fois ce module fonctionnel il suffit d'ajouter à une liste le processus quand il est lancé. Cependant pour le supprimer il a fallu créer un handler sur les signaux `SIGHLD` pour déterminer quand il se terminait (et donc les enlever de la liste). Ceci a pu être efficacement fait grâce aux ressources mises sur moodle. Ci-dessous la structure de la liste et le handler :

```

void suivi_fils (int sig) {

    sig ++;
    int etat_fils, pid_fils;

    do {

```

```

    pid_fils = (int) waitpid(-
1, &etat_fils, WNOHANG | WUNTRACED | WCONTINUED);

    if ((pid_fils == -1) && (errno != ECHILD)) {

        perror("waitpid");

        exit(EXIT_FAILURE);

    } else if (pid_fils > 0) {

        if (WIFSTOPPED(etat_fils)) {

            chgtEtat(pid_fils,SUSPENDU,&tab_jobs);

        } else if (WIFCONTINUED(etat_fils)) {

            chgtEtat(pid_fils,ACTIF,&tab_jobs);

        } else if (WIFEXITED(etat_fils)) {

            supprimer(pid_fils,&tab_jobs);

        } else if (WIFSIGNALED(etat_fils)) {

            /* traître signal */

        }

    }

} while (pid_fils > 0);
/* autres actions après le suivi des changements d'état */
}

```

Mise en place grâce à :

```

signal(SIGCHLD, suivi_fils);

```

Structure de la liste :

```

struct processus {
    int id;
    int pid;
    etat e;
    char * cmd;
    proc suivant;
};

```

La commande stop nécessite de définir l'état d'un processus. Pour cela j'ai dû définir une énumération contenant : SUSPENDUE, ACTIF. Il suffit de lui envoyer le signal SIGSTOP. Grâce à la ligne :

```
kill(get_pid(id,tab_jobs),SIGSTOP);
```

Ensuite grâce au handler définit précédemment il était facile de mettre à jours jobs.

La commande bg n'est pas très compliquée puisqu'il faut envoyer là même ligne que précédemment, mais avec SIGCONT

La commande fg est pareille que bg, mais on rappelle foreground() définit plus haut.

Pour mettre en place ctrlZ j'ai décidé d'opter pour masquer les signaux SIGTSTP pour les fils. Et de les traduire pour le minishell en SIGSTOP. Plus de détail dans la fonction void ctrlZ.

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.

Pour le ctrlC il c'est le même principe que pour ctrlZ, mais en remplaçant SIGINT par SIGKILL cette fois.

Question 8 (Redirections) Compléter votre programme pour permettre d'associer l'entrée standard ou la sortie standard d'une commande à un fichier.

Pour implémenter cette fonctionnalité il a suffi de récupérer les chemins du readcmd par :

```
in = commande->in;
out = commande->out;
```

Ensuite si ils ne sont pas null il suffisait dans un premier temps de les ouvrir et affecter les descripteurs de fichiers:

```
in_desc = xopen(in,O_RDONLY);
out_desc = xopen2(out, O_WRONLY | O_CREAT | O_TRUNC, 0640);
```

A partir de là on affecte le stdin et le stdout avec les lignes suivantes dans les bons fils :

```
int new_in = dup2(in_desc,0); // Premier fils
int dupdesc2 = dup2(out_desc,1); // Dernier fils
```

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

On crée un tube dans le père avec la commande :

```
int retour =pipe(element);
```

Ensuite il faut créer 2 fils avec 2 fork et relier la sortie standard de l'un à l'entrée du tube et l'entrée standard de l'autre à la sortie du tube. On crée donc 2 fork. Dans le fils n°1 (le premier):

```
close(tableau_pipe[num_processus][0]); // Ecrire dans le pipe suivant
int desc = dup2(tableau_pipe[num_processus][1],1); // stdout -> pipe_suisant
```

Ici tableau_pipe est constitué d'un élément les numéros d'entrée/sortie du tube.

Dans le fils n°2 (le dernier) :

```
close(tableau_pipe[num_processus][1]); // lire dans le pipe avant
int desc = dup2(tableau_pipe[num_processus][0],0); // pipe_avant -> stdin
```

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

Cette partie plus délicate à demander une légère restructuration de comment les fils sont créés. J'ai choisi que le père créerait tous les fils et les lierait les tubes entre les fils. Cette méthode n'est pas sans défaut puisqu'il faut que le père ferme tous les pipes. Pour cela on crée déjà les pipes et on stocke les numéros d'entrée sortie dans un tableau.

```
for (num_processus=0; num_processus<nb_commande;num_processus++){
    fflush(stdout);
    pid = fork();
    if (pid <0){
        printf("Erreur Fork");
    }
    if (pid ==0) {
        sigprocmask(SIG_BLOCK,&ens,NULL);
        /* Fermer les pipes inutilise par le fils */
        int j =0;
        while (j<num_processus-1){
            close(tableau_pipe[j][0]);
            close(tableau_pipe[j][1]);
            j++;
        }
        j=num_processus+1;
        while (j<nb_commande-1)
        {
            close(tableau_pipe[j][0]);
            close(tableau_pipe[j][1]);
            j++;
        }

        /* lire le prochain pipe sauf pour le 1er */
        if (num_processus != 0) {
            close(tableau_pipe[num_processus-
1][1]); // lire dans le pipe avant
            int desc = dup2(tableau_pipe[num_processus-
1][0],0); // pipe_avant -> stdin
        } else {
            int new_in = dup2(in_desc,0);
            // ind_desc -> stdin
        }
        /* Ecrire dans le prochain pipe sauf pour le dernier */
        if ( num_processus != nb_commande-1){
            close(tableau_pipe[num_processus][0]);
            // Ecrire dans le pipe suivant
            int desc = dup2(tableau_pipe[num_processus][1],1);
            // stdout -> pipe_suivant
        } else {
```

```

        int dupdesc2 = dup2(out_desc,1);
// stdout -> out_desc
    }
    /* Execution de la commande */
    execvp(seq[num_processus][0],seq[num_processus]);
    perror("Commande not found");
    exit(1);
}

```

On supprime donc les pipes inutilisées par le fils courant crée et on lit sur le tube d'avant et on écrit sur le tube d'après. Sauf pour le premier et dernier.

II Test

Afin de tester les fonctionnalités j'ai fait plusieurs tests :

- Commande cd : on écrit cd ~ il nous renvoie au home normalement :

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI:~$ cd /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ./minishell
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ cd ~
siboudechou@MSI:/home/siboudechou$

```

- Commande ls pour tester les commandes externes :

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI:/home/siboudechou$ exit
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ./minishell
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ls
cd.c cd.o IO.h Makefile minishell minishell.o processus.h readcmd.c readcmd.o
cd.h IO.c IO.o Makefile.bak minishell.c processus.c processus.o readcmd.h toto.c
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$

```

- Commande et &sleep 100 + list + ps v pour vérifier que list marche bien et le background

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ./minishell
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ &sleep 100
siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ list
[1] 486 actif sleep 100

siboudechou@MSI:/mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ps -v
PID TTY STAT TIME MAJFL TRS DRS RSS %MEM COMMAND
188 tty1 S 0:00 0 0 889589444 660 0.0 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" 5763d90
189 tty1 S 0:00 0 0 664486088 716 0.0 sh /mnt/c/Users/leoar/.vscode/extensions/ms-vscode-remote.rem
194 tty1 S 0:00 0 0 735008368 684 0.0 sh /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215
196 tty1 Sl 0:06 0 0 357996332 43300 0.2 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215c
220 tty1 Sl 0:41 0 0 31777436 98796 0.5 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215cb
227 tty1 Sl 0:00 0 0 1032560224 22608 0.1 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215
244 pts/0 Ss 0:00 0 0 819150556 2488 0.0 /bin/bash
274 tty1 Sl 0:06 0 0 940686460 28404 0.1 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppto
311 tty1 Sl 0:00 0 0 1035765900 10104 0.0 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppt
333 tty1 Sl 0:03 0 0 325377336 24732 0.1 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppto
444 tty2 S 0:00 0 0 261799536 2480 0.0 -bash
471 pts/0 S 0:00 0 0 726071276 596 0.0 ./minishell
481 tty2 S 0:00 0 0 156037996 660 0.0 sleep 100
485 tty2 S 0:00 0 0 66390640 624 0.0 ./minishell
486 tty2 S 0:00 0 0 332170044 660 0.0 sleep 100
487 tty2 R 0:00 0 0 859390600 1660 0.0 ps -v

```

- Commande stop et bg 1 pour tester ces commande (sur le même exemple)

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ps
  PID TTY          TIME CMD
  444 tty2      00:00:00 bash
  489 tty2      00:00:00 sleep
  493 tty2      00:00:00 minishell
  494 tty2      00:00:00 sleep
  495 tty2      00:00:00 ps
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ stop 1
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ list
[1] 494 suspendu      sleep 1000

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ps -v
  PID TTY      STAT   TIME   MAJFL   TRS     DRS     RSS %MEM COMMAND
  188 tty1    S      0:00   0       0 889589444 660 0.0 sh -c "$VSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" 5763d90
  189 tty1    S      0:00   0       0 664486088 716 0.0 sh /mnt/c/Users/leoar/.vscode/extensions/ms-vscode-remote.rem
  194 tty1    S      0:00   0       0 735008368 684 0.0 sh /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215
  196 tty1    Sl     0:07   0       0 357996844 43420 0.2 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215c
  220 tty1    Sl     0:44   0       0 31778556 100140 0.6 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215c
  227 tty1    Sl     0:00   0       0 1032560224 22492 0.1 /home/siboudechou/.vscode-server/bin/5763d909d5f12fe19f215
  244 pts/0   Ss     0:00   0       0 819150556 2488 0.0 /bin/bash
  274 tty1    Sl     0:07   0       0 940686460 28288 0.1 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppto
  311 tty1    Sl     0:00   0       0 1035765900 10104 0.0 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppt
  333 tty1    Sl     0:03   0       0 325377336 24672 0.1 /home/siboudechou/.vscode-server/extensions/ms-vscode.cppto
  444 tty2    S      0:00   0       0 261799536 2480 0.0 -bash
  471 pts/0   S      0:00   0       0 726071276 596 0.0 ./minishell
  489 tty2    S      0:00   0       0 385110480 656 0.0 sleep 1000
  493 tty2    S      0:00   0       0 925659220 636 0.0 ./minishell
  494 tty2    T      0:00   0       0 572494068 656 0.0 sleep 1000
  496 tty2    R      0:00   0       0 646982048 1652 0.0 ps -v

```

- ctrlC ctrl Z j'ai ouvert un 2 eme terminal pour vérifier l'état suspendu du processus

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ./minishell
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ sleep 12345
^Csiboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ sleep 12345
^Z

siboudechou@MSI: ~
  649 tty2    T      0:00   0       0 356190780 660 0.0 sleep 12345
  650 tty3    R      0:00   0       0 700324208 1656 0.0 ps -v
siboudechou@MSI: ~$

```

- Commande cat < f1 > f2 pour tester les redirections

- Commande cat toto.c | grep int | wc -l (fichier toto.c fourni) pour tester les pipelines

```

siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ ./minishell
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$ cat toto.c | grep int | wc -l
4
siboudechou@MSI: /mnt/c/Users/leoar/Documents/n7/Sec/Projet/Archive/Source$

```