



Systèmes d'Exploitations Centralisés

(mini)Projet (mini)Shell

Rendu Intermediaire

1ère Année, Département Sciences du Numérique

Younes Saoudi

2019-2020

Contents

1	Rendu Intermediaire	2
2	Code Source	4
2.1	Question 1	4
2.2	Question 2	5
2.3	Question 3	6
2.4	Question 4	7
2.5	Question 5	9

Rendu Intermediaire

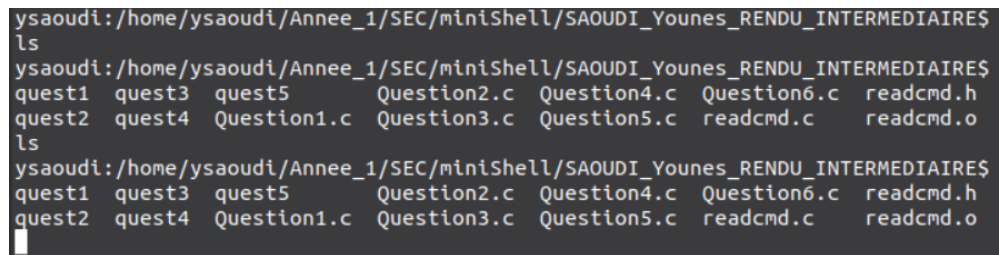
Question 1

Réalisation de 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.

L'implantation de cette question était assez simple: Il fallait d'abord une boucle infinie (`while(1)`) au sein de laquelle la création des processus était exécutée. En testant le retour des appels systèmes, et en ajoutant, dans la partie du fils, les commandes

```
execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
```

Question 2



```
ysaoudi:/home/ysaoudi/Annee_1/SEC/miniShell/SAOUDI_Younes_RENDU_INTERMEDIAIRES$
ls
ysaoudi:/home/ysaoudi/Annee_1/SEC/miniShell/SAOUDI_Younes_RENDU_INTERMEDIAIRES$
quest1 quest3 quest5 Question2.c Question4.c Question6.c readcmd.h
quest2 quest4 Question1.c Question3.c Question5.c readcmd.c readcmd.o
ls
ysaoudi:/home/ysaoudi/Annee_1/SEC/miniShell/SAOUDI_Younes_RENDU_INTERMEDIAIRES$
quest1 quest3 quest5 Question2.c Question4.c Question6.c readcmd.h
quest2 quest4 Question1.c Question3.c Question5.c readcmd.c readcmd.o
```

(a) Question 2

Figure 1.1: Mise en exergue du fait que l'affichage de l'invite se mêle à l'exécution du processus fils.

L'ajout des commandes suivantes

```
printf("ysaoudi:");
printf("%s\n", getcwd(s,200));
```

avant le `fork()` engendre le problème remarqué dans la figure ci-dessus.

Question 3

Modification du code afin qu'il attende la fin de la dernière commande lancée avant de passer à la lecture de la ligne suivante.

Il suffisait d'ajouter `wait(NULL)` dans la partie exécutée par le père.

Question 4

Ajout de deux commandes internes, exécutées directement par l'interpréteur sans lancer de processus fils : `cd` et `exit`.

Pour ce faire, il fallait contrôler la saisie de ces commandes avant même la création du processus fils avec une simple structure `if.. else if...`

Question 5

Ajout de la possibilité du lancement de commandes en tâche de fond, spécifié par un `&` en fin de ligne. Pour achever cela, il fallait n'exécuter `wait(NULL)` que si l'attribut `backgrounded` de la structure `cmdline` était `NULL`, i.e., si la commande n'était pas en tâche de fond.

Questions 6 et 7

Je n'ai malheureusement pas eu la chance d'implanter les questions 6 et 7 par manque de temps et de compréhension des compétences nécessaires à l'implémentation.

Code Source

2.1 Question 1

```
1 #include <stdio.h> /* entr es sorties */
2 #include <unistd.h> /* pimitives de base : fork, ...*/
3 #include <stdlib.h> /* exit */
4 #include <signal.h> /* traitement des signaux */
5 #include <sys/wait.h>
6 #include "readcmd.h"
7
8 /**
9  * @author Younes SAOUDI 1SN-D
10  * Question 1
11  * R aliser la boucle de base de l'interpr teur , en se limitant des commandes simples (
12  * pas d'op rateurs de composition), sans motifs pour les noms de fichiers.
13  */
14 int main()
15 {
16     int retour;
17     struct cmdline *cmd;
18
19     while (1)
20     {
21         cmd = readcmd();
22         retour = fork();
23
24         /* Bonne pratique : tester syst matiquement le retour des appels syst me */
25         if (retour < 0)
26         { /* chec du fork */
27             printf("Erreur fork\n");
28             /* Convention : s'arr ter avec une valeur > 0 en cas d'erreur */
29             exit(1);
30         }
31
32         /* fils */
33         if (retour == 0)
34         {
35             execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
36             exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
37         }
38
39         /* pere */
40         else
41         {
42             //Ne rien faire
43         }
44     }
45     return EXIT_SUCCESS;
46 }
```

2.2 Question 2

```
1 #include <stdio.h> /* entr es sorties */
2 #include <unistd.h> /* pimitives de base : fork, ...*/
3 #include <stdlib.h> /* exit */
4 #include <signal.h> /* traitement des signaux */
5 #include "readcmd.h"
6 #include <sys/wait.h>
7
8 /**
9  * @author Younes SAOUDI 1SN-D
10  * Question 2
11  * Construire une session simple (utilisant le code crit pour la question 1) mettant en
12  * vidence le fait que l'affichage de l'invite pr c de ou se m le
13  * l'ex cution du processus fils.
14  */
15 int main()
16 {
17     char s[200];
18     int retour;
19     struct cmdline *cmd;
20
21     while (1)
22     {
23         //MODIFICATION QUESTION 2
24         printf("ysaoudi:");
25         printf("%s$\n", getcwd(s,200));
26         //FIN QUESTION 2
27         cmd = readcmd();
28         retour = fork();
29
30         /* Bonne pratique : tester syst matiquement le retour des appels syst me */
31         if (retour < 0)
32         { /* chec du fork */
33             printf("Erreur fork\n");
34             /* Convention : s'arr ter avec une valeur > 0 en cas d'erreur */
35             exit(1);
36         }
37
38         /* fils */
39         if (retour == 0)
40         {
41             execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
42             exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
43         }
44
45         /* pere */
46         else
47         {
48             //Ne rien faire
49         }
50     }
51     return EXIT_SUCCESS;
52 }
```

2.3 Question 3

```
1 #include <stdio.h> /* entr es sorties */
2 #include <unistd.h> /* pimitives de base : fork, ...*/
3 #include <stdlib.h> /* exit */
4 #include <signal.h> /* traitement des signaux */
5 #include "readcmd.h"
6 #include <sys/wait.h>
7
8 /**
9  * @author Younes SAOUDI 1SN-D
10  * Question 3
11  * Modifier le code afin qu'il attende la fin de la derni re commande lanc e avant de
12  * passer la lecture de la ligne suivante.
13  */
14 int main()
15 {
16     char s[200];
17     int retour;
18     struct cmdline *cmd;
19
20     while (1)
21     {
22         //MODIFICATION QUESTION 2
23         printf("ysaoudi:");
24         printf("%s$\n", getcwd(s,200));
25         //FIN QUESTION 2
26         cmd = readcmd();
27         retour = fork();
28
29         /* Bonne pratique : tester syst matiquement le retour des appels syst me */
30         if (retour < 0)
31         { /* chec du fork */
32             printf("Erreur fork\n");
33             /* Convention : s'arr ter avec une valeur > 0 en cas d'erreur */
34             exit(1);
35         }
36
37         /* fils */
38         if (retour == 0)
39         {
40             execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
41             exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
42         }
43
44         /* pere */
45         else
46         {
47             //MODIFICATION QUESTION 3
48             wait(NULL);
49             //FIN QUESTION 3
50         }
51     }
52     return EXIT_SUCCESS;
53 }
```


2.4 Question 4

```
1 #include <stdio.h> /* entr es sorties */
2 #include <unistd.h> /* pimitives de base : fork, ...*/
3 #include <stdlib.h> /* exit */
4 #include <signal.h> /* traitement des signaux */
5 #include "readcmd.h"
6 #include <sys/wait.h>
7
8 /**
9  * @author Younes SAOUDI 1SN-D
10  * Question 4
11  * Compl ter le code en ajoutant deux commandes internes, ex cut es directement par l'
12  * interpr teur sans lancer de processus fils : cd et exit.
13  */
14 int main()
15 {
16     char s[200];
17     int retour;
18     struct cmdline *cmd;
19
20     while (1)
21     {
22         //MODIFICATION QUESTION 2
23         printf("ysaoudi:");
24         printf("%s$\n", getcwd(s,200));
25         //FIN QUESTION 2
26         cmd = readcmd();
27
28         //MODIFICATION QUESTION 4
29         if (strcmp(cmd->seq[0][0], "cd") == 0)
30         {
31             chdir(cmd->seq[0][1]);
32             continue;
33         }
34         else if (strcmp(cmd->seq[0][0], "exit") == 0)
35         {
36             exit(0);
37         }
38         //FIN QUESTION 4
39
40         retour = fork();
41
42
43         /* Bonne pratique : tester syst matiquement le retour des appels syst me */
44         if (retour < 0)
45         { /* chec du fork */
46             printf("Erreur fork\n");
47             /* Convention : s'arr ter avec une valeur > 0 en cas d'erreur */
48             exit(1);
49         }
50
51         /* fils */
52         if (retour == 0)
53         {
54             execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
55             exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
56         }
57
58         /* pere */
59         else
60         {
61             //MODIFICATION QUESTION 3
```

```
62         wait(NULL);
63         //FIN QUESTION 3
64     }
65 }
66 return EXIT_SUCCESS;
67 }
```

2.5 Question 5

```
1 #include <stdio.h> /* entr es sorties */
2 #include <unistd.h> /* pimitives de base : fork, ...*/
3 #include <stdlib.h> /* exit */
4 #include <signal.h> /* traitement des signaux */
5 #include "readcmd.h"
6 #include <sys/wait.h>
7
8 /**
9  * @author Younes SAOUDI 1SN-D
10  * Question 5
11  * Ajouter la possibilit e de lancement de commandes en t che de fond, sp cifi e par un &
12  * en fin de ligne.
13  */
14 int main()
15 {
16     char s[200];
17     int retour;
18     struct cmdline *cmd;
19
20     while (1)
21     {
22         //MODIFICATION QUESTION 2
23         printf("ysaoudi:");
24         printf("%s$\n", getcwd(s,200));
25         //FIN QUESTION 2
26         cmd = readcmd();
27
28         //MODIFICATION QUESTION 4
29         if (strcmp(cmd->seq[0][0], "cd") == 0)
30         {
31             chdir(cmd->seq[0][1]);
32             continue;
33         }
34         else if (strcmp(cmd->seq[0][0], "exit") == 0)
35         {
36             exit(0);
37         }
38         //FIN QUESTION 4
39
40         retour = fork();
41
42         /* Bonne pratique : tester syst matiquement le retour des appels syst me */
43         if (retour < 0)
44         { /* chec du fork */
45             printf("Erreur fork\n");
46             /* Convention : s'arr ter avec une valeur > 0 en cas d'erreur */
47             exit(1);
48         }
49
50         /* fils */
51         if (retour == 0)
52         {
53             execvp(cmd->seq[0][0], cmd->seq[0]); //Executer la commande
54             exit(EXIT_SUCCESS); /* Terminaison normale (0 = sans erreur) */
55         }
56
57         /* pere */
58         else
59         {
60             //MODIFICATION QUESTION 5
61             if (cmd->backgrounded == NULL){
62                 wait(NULL);
63             }
64         }
65     }
66 }
```

```
62         } else {  
63             //RIEN  
64         }  
65         //FIN QUESTION 5  
66     }  
67 }  
68 return EXIT_SUCCESS;  
69 }
```