



Systèmes d'exploitation centralisés

Rapport : minishell

1SN-F

Élève :

Louis Thevenet

Enseignant :

Emmanuel Chaput

4 Mai 2024

Table des matières

1. Gestion des processus	3
1.1. Enchaînement séquentiel des commandes	3
1.2. Exécution en arrière-plan	3
2. Gestion des tâches	3
3. Signaux	3
3.1. Signal SIGCHLD	3
3.2. Signaux SIGINT, SIGTSTP	4
4. Fichiers et redirections	5
5. Tubes	7

1. Gestion des processus

1.1. Enchaînement séquentiel des commandes

```
1 > ls
2 flake.lock flake.nix projet rapport result sujets tp
3 > echo test
4 test
5 > cat ./projet/test
6 exemple
7 > exit
8 Au revoir ...
```

Liste 1. – On attend la fin de l'exécution du fils pour passer à la prochaine commande

1.2. Exécution en arrière-plan

```
1 > cat ./projet/test
2 #!/usr/bin/env bash
3
4 sleep 5
5 echo "Done!"
6 > ./projet/test &
7 >
8 > ls
9 flake.lock flake.nix projet rapport result sujets tp
10 > Done!
11 >
```

Liste 2. – On exécute la commande en arrière-plan

On peut également vérifier la bonne terminaison du fils après exécution via `watch ps -sf` :

```
1 S+ pts/2 0:00 \_ /nix/store/ccj0aml5v6xdvkqrvq-minishell/bin/minishell
2 S+ pts/2 0:00 \_ bash ./projet/test
3 S+ pts/2 0:00 \_ sleep 5
```

Liste 3. – Durant l'exécution

```
1 S+ pts/2 0:00 \_ /nix/store/ccj0aml5v6xdvkqrvq-minishell/bin/minishell
```

Liste 4. – Après exécution

2. Gestion des tâches

Comme proposé dans l'énoncé du projet, on ajoute des commandes internes au minishell :

lj (list jobs) Affiche les tâches en cours

sj <id> (stop job) Arrête la tâche d'identifiant `id` (envoie `SIGSTOP`)

fg <id> (foreground) Met la tâche d'identifiant `id` en avant-plan

bg <id> (background) Met la tâche d'identifiant `id` en arrière-plan

Ces commandes seront utilisées dans la suite pour illustrer l'état du minishell.

3. Signaux

3.1. Signal `SIGCHLD`

Un mode `debug` a été ajouté au projet afin d'afficher des informations sur les signaux reçus.

Dans la Liste 5, on :

- attend normalement la fin d'exécution de la commande en avant-plan
- attend normalement la fin d'exécution de la commande en arrière-plan
- envoie le signal `SIGCHLD` au processus fils
- envoie le signal `SIGSTOP` au processus fils
- envoie le signal `SIGCONT` au processus fils
- affiche les jobs en cours pour constater que le fils est continué en arrière-plan
- On teste ensuite les signaux `SIGSTOP` et `SIGCONT` sur un job en arrière-plan.

```

1  > sleep 2
2  [Child 294501 exited]
3  > sleep 2&
4  > [Child 294709 exited]
5
6  > sleep 9999
7  [Child 296018 signaled]
8  > sleep 9999
9  [Child 299625 stopped]
10 > [Child 299625 continued]
11
12 > lj
13 Job 0
14   PID: 299625
15   STDOUT -> 4207432  State: Active  Command: sleep 9999
16 >
17 >
18 >
19 >
20 >
21 > sleep 999&
22 > [Child 335139 stopped]
23 [Child 335139 continued]
24
25 > lj
26 Job 0
27   PID: 299625
28   STDOUT -> 4207432  State: Active  Command: sleep 9999
29
30 Job 1
31   PID: 335139
32   STDOUT -> 4207432  State: Active  Command: sleep 999
33 >

```

Liste 5. – Démonstration `SIGCHLD`

3.2. Signaux `SIGINT`, `SIGTSTP`

On voit dans cet exemple que le programme père reçoit le signal `SIGINT`, qu'il décide de tuer le fils en avant-plan, finalement le message informant la terminaison du processus fils est affiché.

```

1  > sleep 10
2  ^C[SIGINT received]
3  [Killing 343628]
4  > [Child 343628 signaled]
5
6  >

```

Liste 6. – Interruption au clavier

```

1  > sleep 10
2  ^Z[SIGTSTP received]
3  [Stopping 348897]
4  > [Child 348897 stopped]
5
6  > lj
7  Job 0
8      PID: 348897
9  STDOUT -> 4207432  State: Suspended  Command: sleep 10
10 >

```

Liste 7. – Envoie du signal SIGTSTP

Les processus fils quant à eux ne reçoivent pas les signaux SIGINT et SIGTSTP, c'est le père qui gère les interruptions.

```

1  > sleep 999&
2  > ^C[SIGINT received]
3  ^C[SIGINT received]
4  ^Z[SIGTSTP received]
5  ^Z[SIGTSTP received]
6
7  > lj
8  Job 0
9      PID: 355672
10 STDOUT -> 4207432  State: Active  Command: sleep 999
11 >

```

Liste 8. – Les processus fils masquent les signaux SIGINT et SIGTSTP

4. Fichiers et redirections

Le mode debug affiche les nombres d'octets lus et écrits.

```

1  > cat < ./projet/test
2  [read 42, wrote 42]
3  [read 0, wrote 0]
4  [Child 137265 exited]
5  #!/usr/bin/env bash
6
7  sleep 5
8  echo "Done!"
9  [Child 137264 exited]
10 > cat < ./projet/test > test
11 [read 42, wrote 42]
12 [read 0, wrote 0]
13 [Child 137384 exited]
14 [read 42, wrote 42]
15 [read 0, wrote 0]
16 [Child 137383 exited]
17 > [Child 137385 exited]
18
19 > cat test
20 #!/usr/bin/env bash
21
22 sleep 5
23 echo "Done!"
24 [Child 137434 exited]

```

Liste 9. – Démonstration des redirections vers des fichiers en avant plan

L'exemple suivant montre que les redirections continuent de fonctionner en arrière-plan et respectent l'état du processus qui les envoie ou reçoit. On lance en arrière-plan un script dont la sortie est redirigée vers un fichier, on peut le mettre en pause et le reprendre sans que la redirection ne s'arrête.

```
1 > cat ./projet/test_boucle
2 #!/usr/bin/env bash
3
4 for i in {1..50}; do
5     echo "Iteration $i"
6     sleep 3
7 done
8 echo "Done!"
9 > ./projet/test_boucle > sortie &
10 > cat sortie
11 Iteration 1
12 Iteration 2
13 > #on attend un peu
14 Error: command failed to execute
15 > cat sortie
16 Iteration 1
17 Iteration 2
18 Iteration 3
19 Iteration 4
20 > lj
21 Job 0
22   PID: 153453
23   STDOUT -> 4 State: Active Command: ./projet/test_boucle
24 > sj 0
25 > lj
26 Job 0
27   PID: 153453
28   STDOUT -> 4 State: Suspended Command: ./projet/test_boucle
29 > cat sortie
30 Iteration 1
31 Iteration 2
32 Iteration 3
33 Iteration 4
34 Iteration 5
35 Iteration 6
36 Iteration 7
37 > cat sortie
38 Iteration 1
39 Iteration 2
40 Iteration 3
41 Iteration 4
42 Iteration 5
43 Iteration 6
44 Iteration 7
45 > bg 0
46 > cat sortie
47 Iteration 1
48 Iteration 2
49 Iteration 3
50 Iteration 4
51 Iteration 5
52 Iteration 6
53 Iteration 7
54 Iteration 8
55 Iteration 9
```

Liste 10. – Démonstration des redirections vers des fichiers en arrière-plan (sans affichage debug)

5. Tubes

```
1 > ls | wc
2 [Child 176273 exited]
3      9      9     62
4 [Child 176274 exited]
5 >
6 > cat flake.nix | grep pkgs | wc -l
7 [Child 178832 exited]
8 [Child 178833 exited]
9 7
10 [Child 178834 exited]
11 >
```

Liste 11. – Démonstration de tubes en avant-plan

L'exemple suivant montre que l'information du descripteur de fichier vers lequel la sortie est redirigée est stockée dans la liste des jobs.

```
1 > ./projet/test_boucle | wc &
2 > lj
3 Job 0
4   PID: 179824
5   STDOUT -> 4207432   State: Active   Command: ./projet/test_boucle
6 Job 1
7   PID: 179825
8   STDOUT -> 12800864   State: Active   Command: wc
9 >      11      11     27
10 [Child 179824 exited]
11 [Child 179825 exited]
```

Liste 12. – Démonstration de tubes en arrière-plan