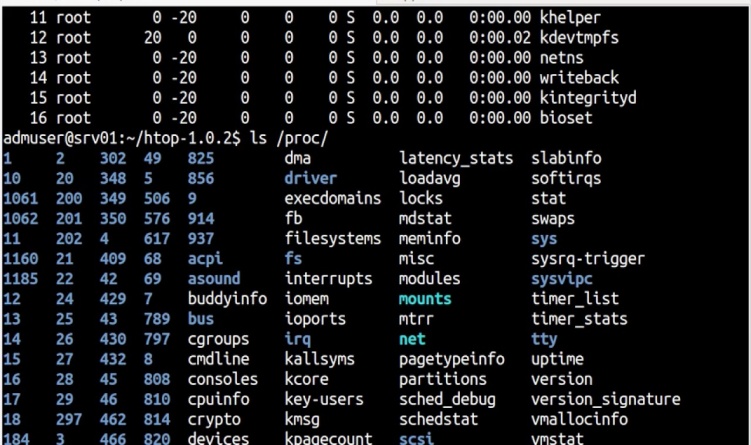
Raphaël CASTANIER

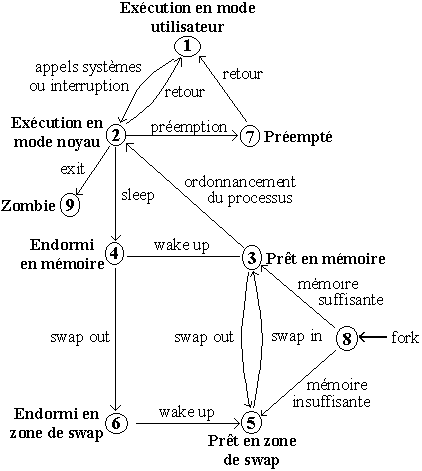
Joffrey RIUS BERNAL

Groupe TP2/ OS308

Compte Rendu TP3

Gestion des processus





Année Académique 2017/2018

Sommaire

[TP3 - Gestion des processus 2](#_Toc510682227)

[1) Contrôle des processus 2](#_Toc510682228)

[1.1 Travaux en arrière-plan (backgrounding) 2](#_Toc510682229)

[1.1 Travaux en premier plan (Foregrounding) 3](#_Toc510682230)

[1.2 Commande PS 3](#_Toc510682231)

[1.4 Commande KILL 4](#_Toc510682232)

[1.5 Commande TOP 4](#_Toc510682233)

[2) Création de processus (fonction fork) 5](#_Toc510682234)

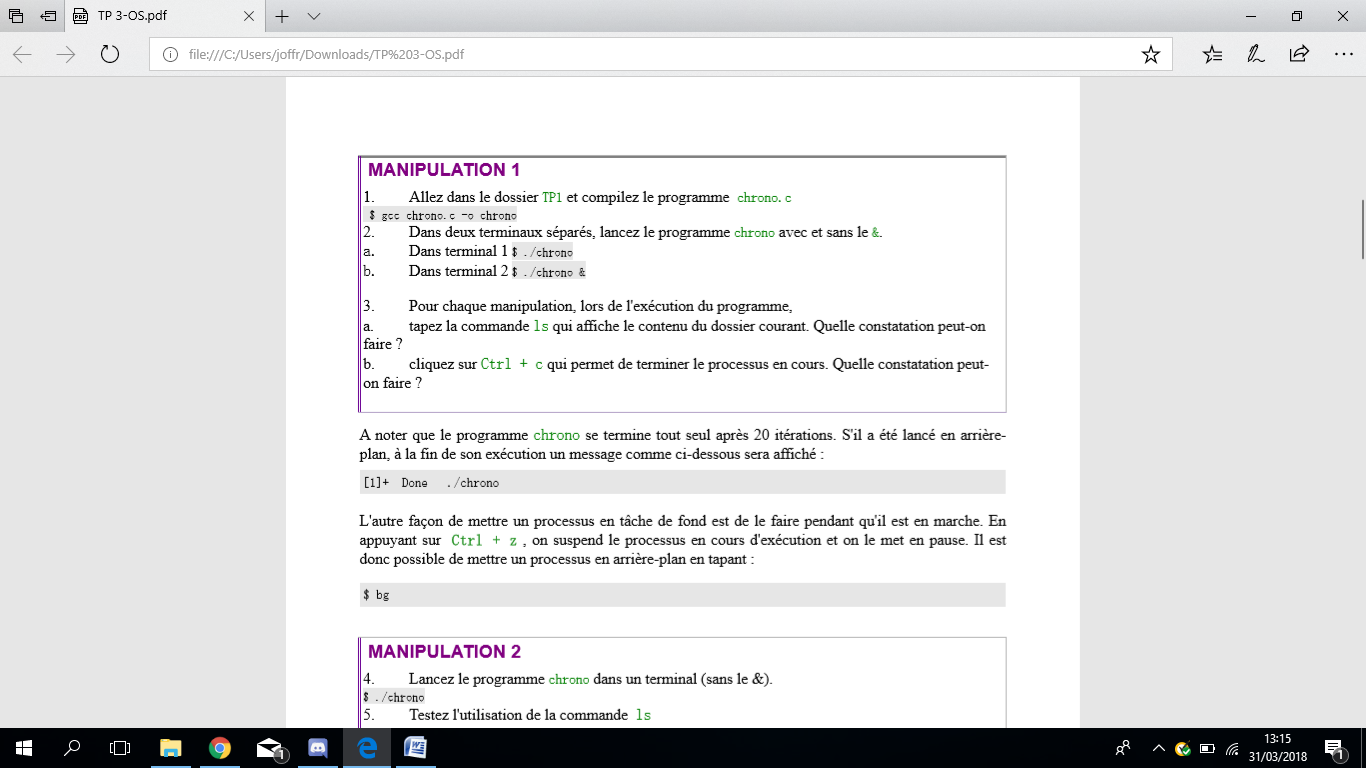
[Exercice 1 5](#_Toc510682235)

[Exercice 2 5](#_Toc510682236)

# TP3 - Gestion des processus

## Contrôle des processus

### Travaux en arrière-plan (backgrounding)

**Manipulation 1**

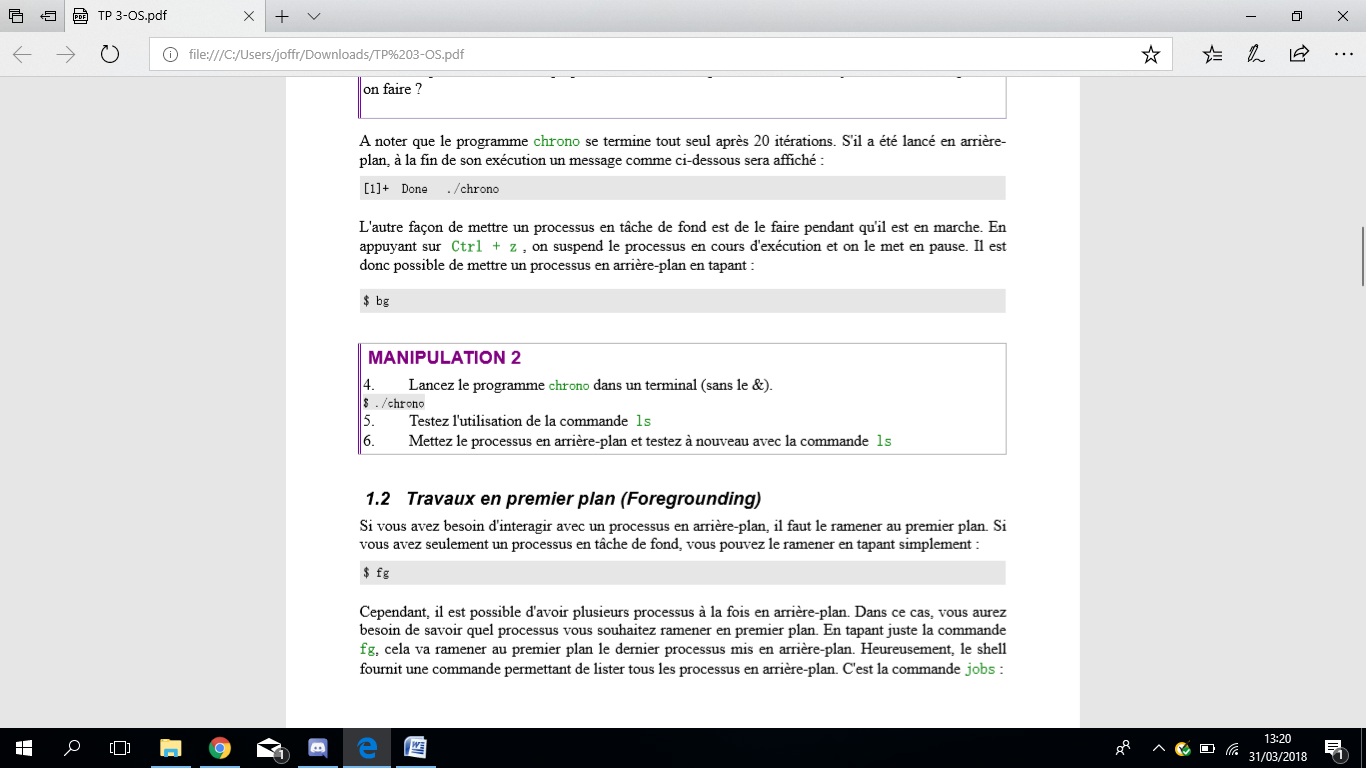
Dans terminal 1 (Sans &) :

La commande ’ls’ ne s’affiche pas pendant que `chrono` tourne mais apparait après son exécution. Pour terminer le processus étant donné que le programme est en 1er plan la commande ‘Ctrl +C’ arrête le processus.

Dans terminal 2 (Avec &) :

La commande ‘ls’ fonctionne simultanément avec l’exécution de ‘chrono car le `&` a lancé son exécution en tâche de fond ce qui libère la main au Shell pour lancer plusieurs commandes. Etant donné que l’esperluette (&) a mis le programme en arrière plan, ‘chrono’ s’exécute jusqu'à son nombre d’itération max.

**Manipulation 2**



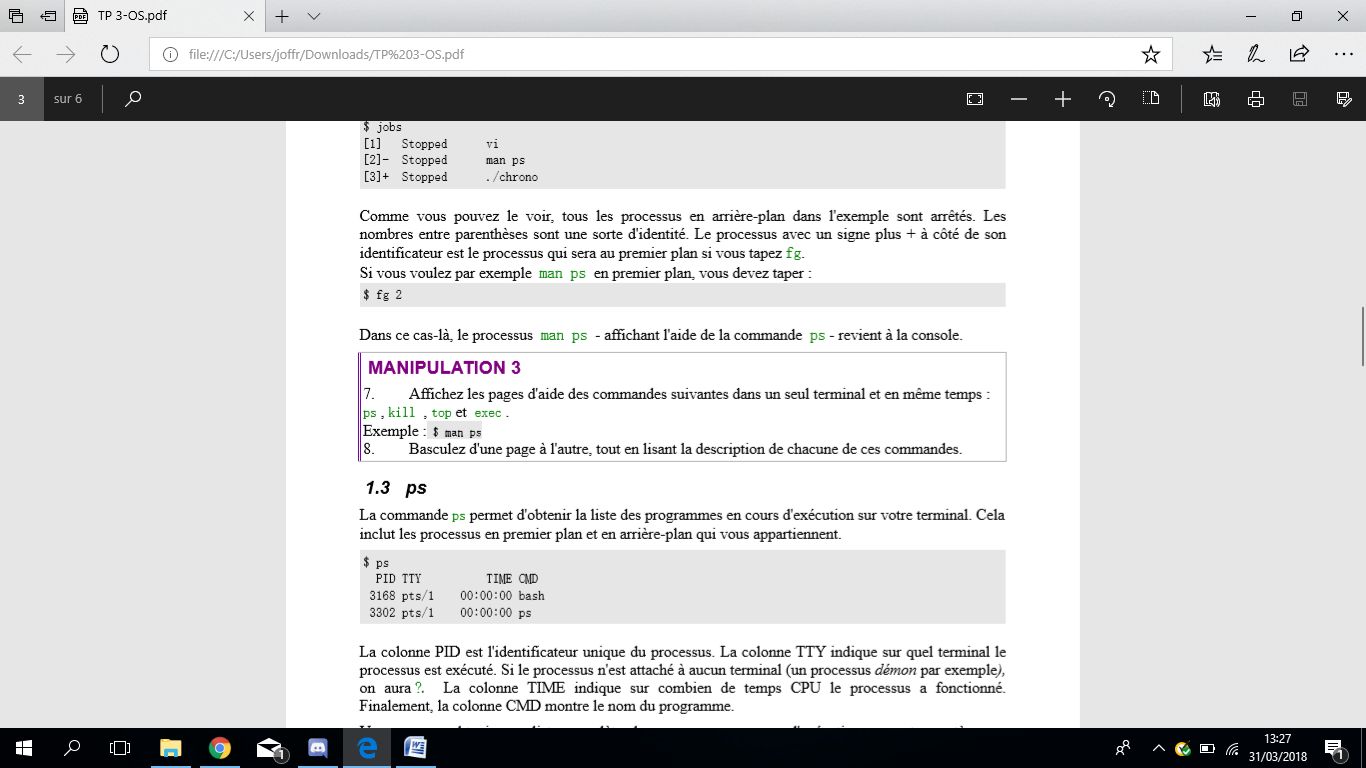
Méthode :

1. `ls` ne fonctionne pas.
2. `CTRL+Z` met bien en arrêt.
3. `bg` relance en arrière plan.
4. `ls` fonctionne

En utilisant cette méthode, nous reprenons les caractéristiques de la manipulation N°1.

### Travaux en premier plan (Foregrounding)

**Manipulation 3**



Afin de lancer plusieurs ‘man’ nous avons saisi la commande ci-dessous :

`man ps & man kill & man top & man exec &`

Ensuite, pour lister les processus et pouvoir vérifier les numéros d’indice afin d’ouvrir la page voulue, nous avons saisi cette commande :

`jobs`

Une fois la liste affichée, nous pouvons sélectionner le processus à mettre en premier plan en écrivant :

`fg <numero\_processus>`

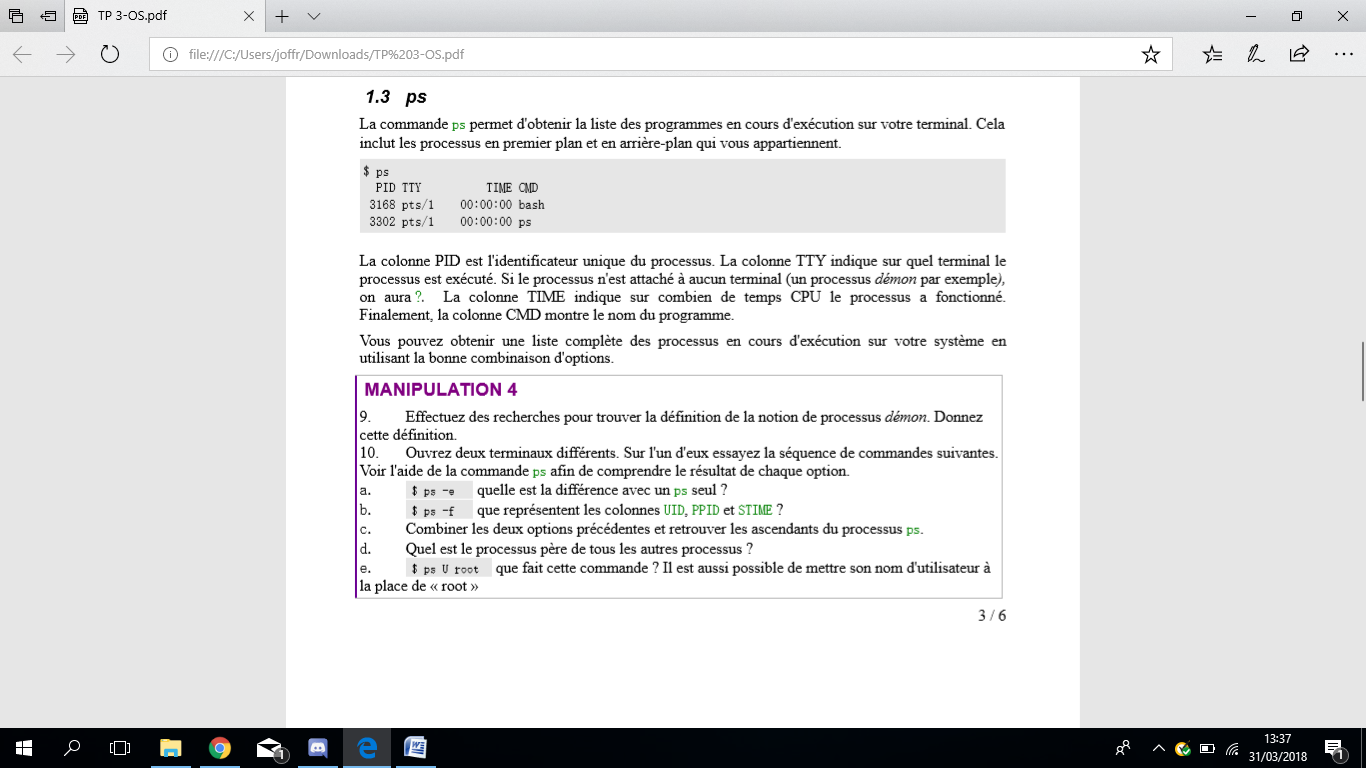
Puis pour arrêter ce processus :

`CTRL+Z`

Ce fonctionnement nous permet donc de créer plusieurs processus puis de sélectionner celui que l’on veut mettre en 1er plan (Foregrounding).

### Commande PS

**Manipulation 4**



Processus démon : Processus qui n'a pas d'interaction avec l'utilisateur, il est fils du processus de `PID=1`. Un processus devient un démon lorsque son père est tué. Le meilleur moyen de créer un démon est donc de tuer le processus du père.

`ps` : affiche la liste des processus de l'utilisateur

`ps -e` : affiche la liste de tous les processus

`UID` : identifiant de l'utilisateur qui a lancé ce processus

`PPID` : identifiant du processus père

`STIME` : temps système du processus

Le processus père de tous les autres est `init` (1).

`ps U root` : affiche les processus de l'utilisateur `root`

### 1.4 Commande KILL

Arrêter un processus en connaissant son PID (Envoi du signal SIGTERM)

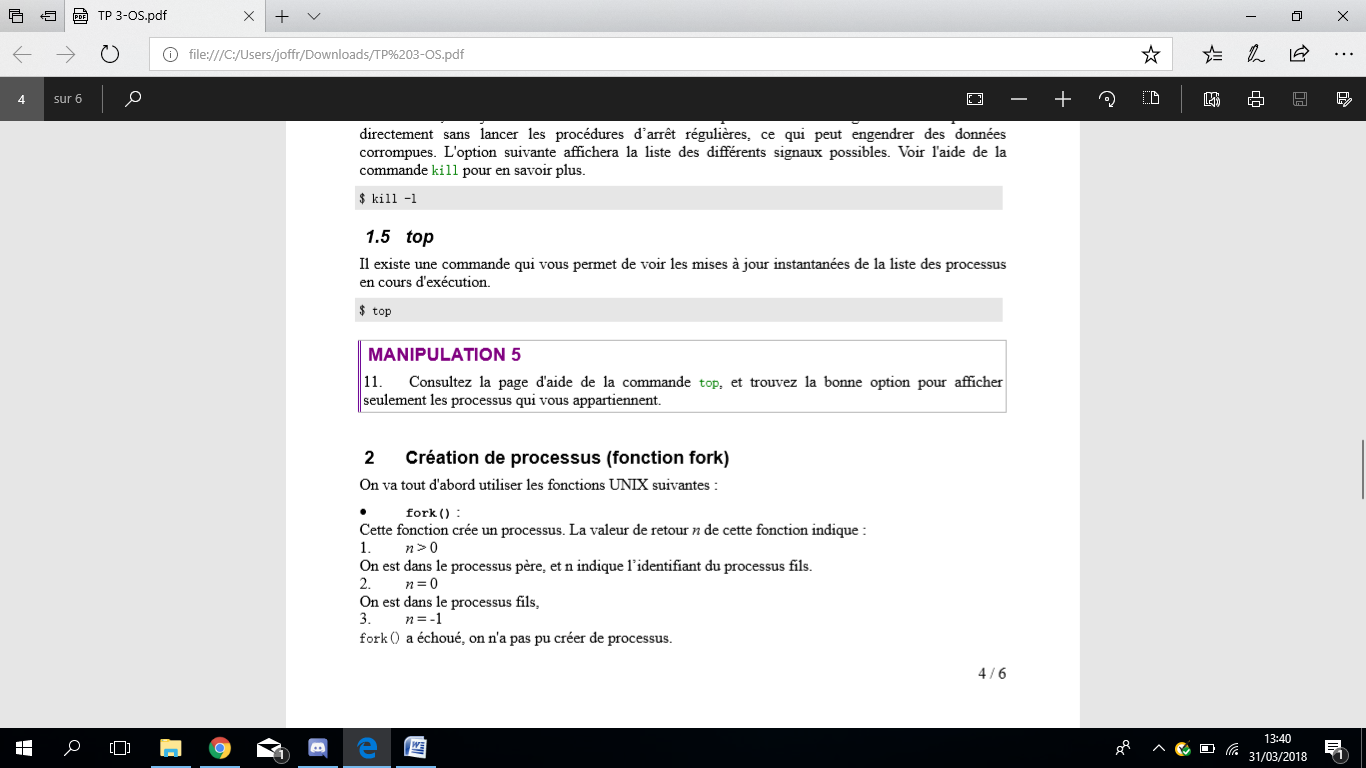
$ kill +N° PID

Arrêter un processus de façon non régulière (Envoi du signal SIGKILL)

$ kill -9 +N° PID

### 1.5 Commande TOP

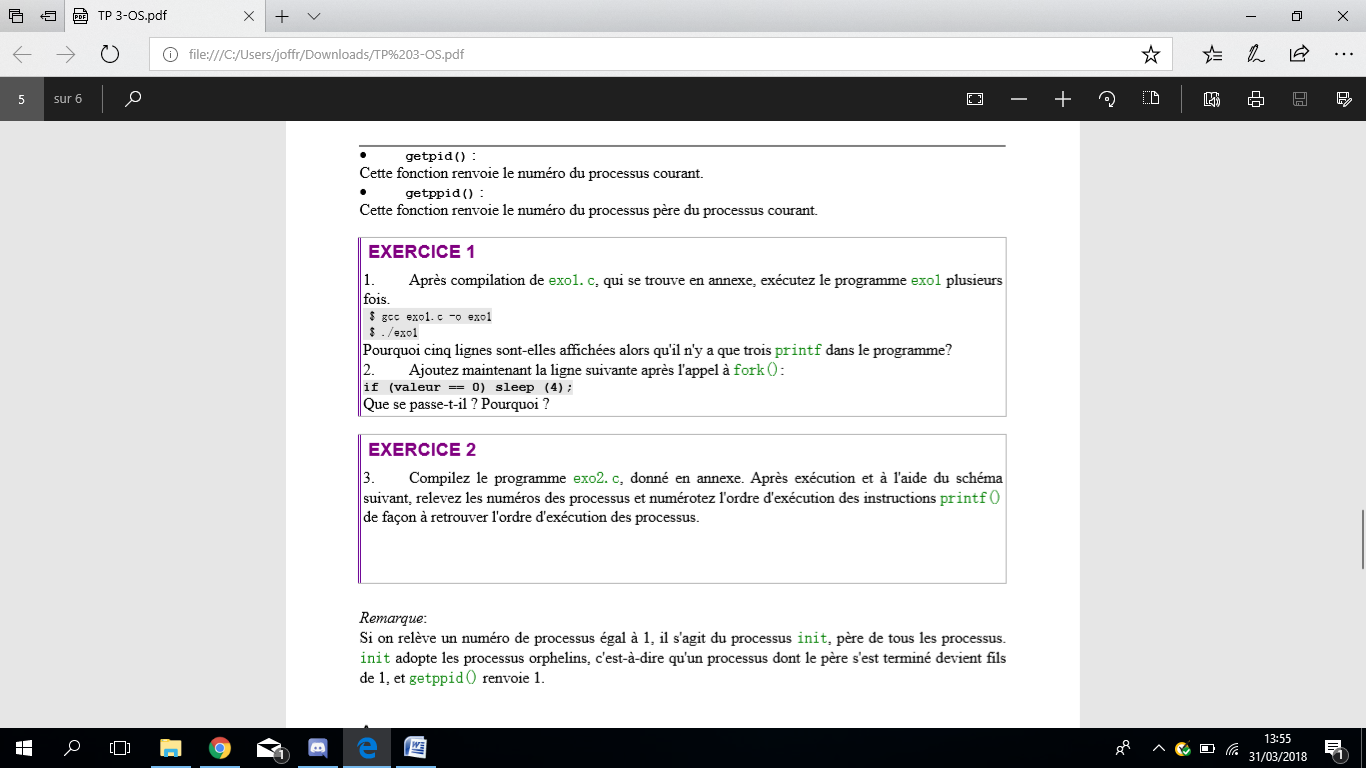
**Manipulation 5**

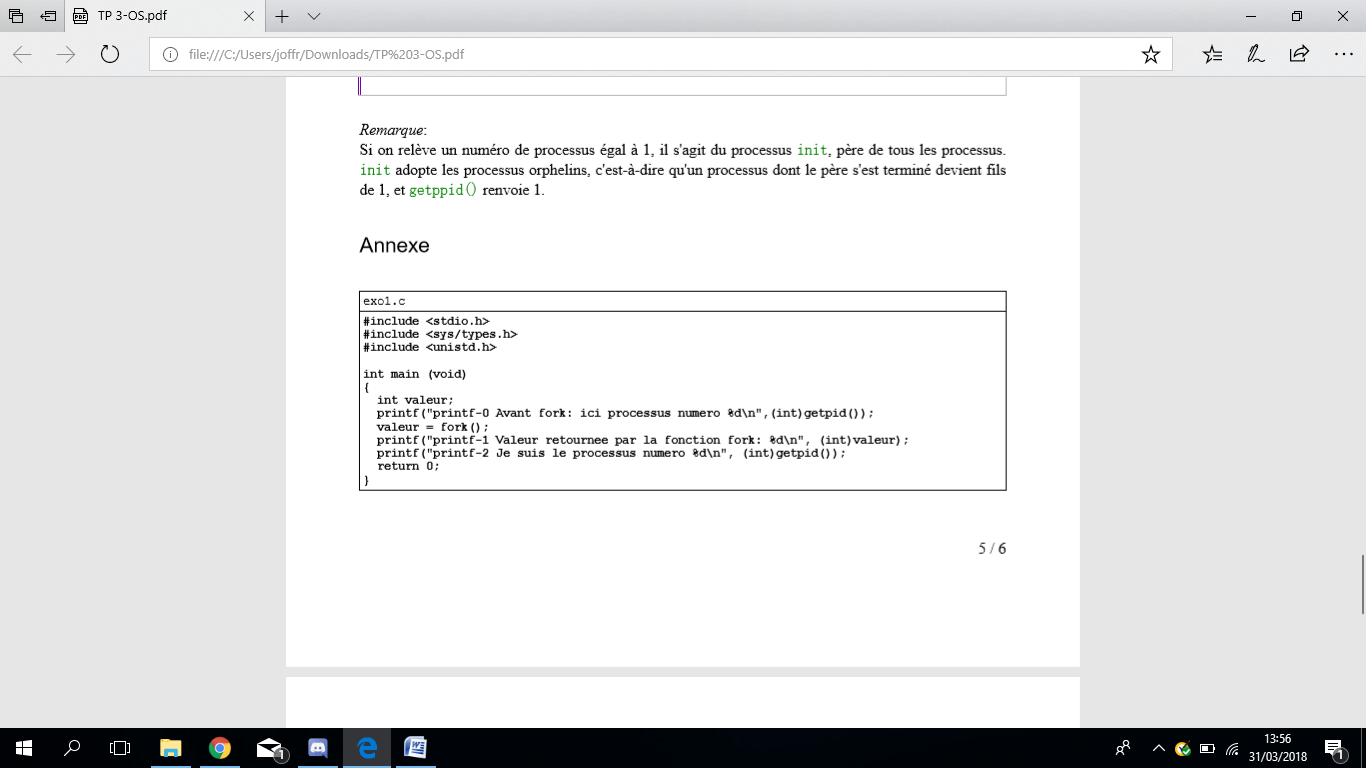


`top -u user` : affiche seulement les processus de l'utilisateur `user`

## Création de processus (fonction fork)

## Exercice 1





1. Lors de l'exécution de `exo1`, le premier `printf` est exécuté, puis `fork()` est appelé.

Ensuite, le père \*\*ET\*\* le fils exécute les deux autres `printf`.

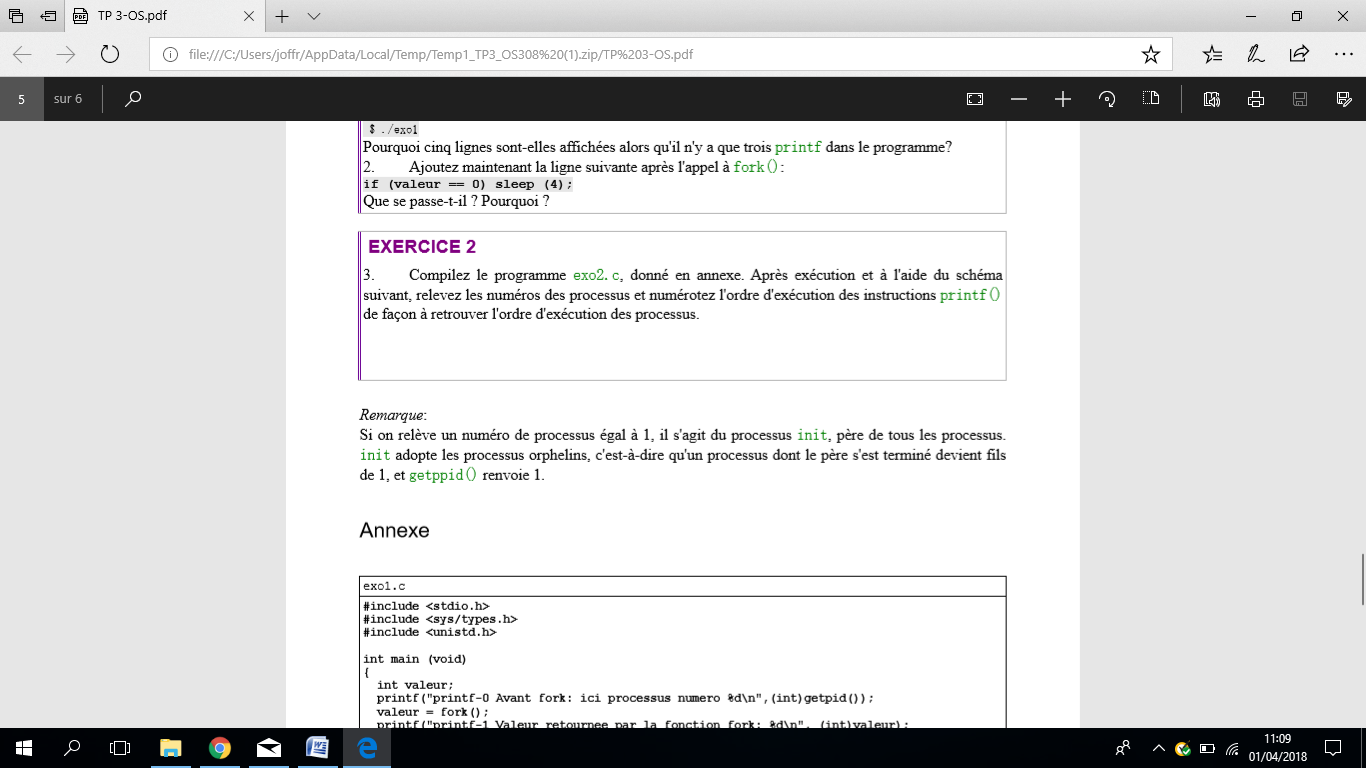
Au total, 5 `printf` sont affichés à l'écran.

1. Avec `if (valeur == 0) sleep (4);`, le fils attend 4 secondes avant d'exécuter les 2 `printf`du fils

Cela permet de voir l’évolution des ‘printf’ et de pouvoir mieux comprendre le fonctionnement.

Le ‘if’ sélectionne uniquement le fils car il teste la valeur de retour du ‘fork()’ égale à 0 donc la valeur du processus fils.

## Exercice 2



et exécuté le résultat suivant, qui nous a permis de dessiner leci-dessous

Retour de exo2

PID PPID INSTRUCTION RETOUR-FORK

2293 2 print 1

2293 2 print 2 2294

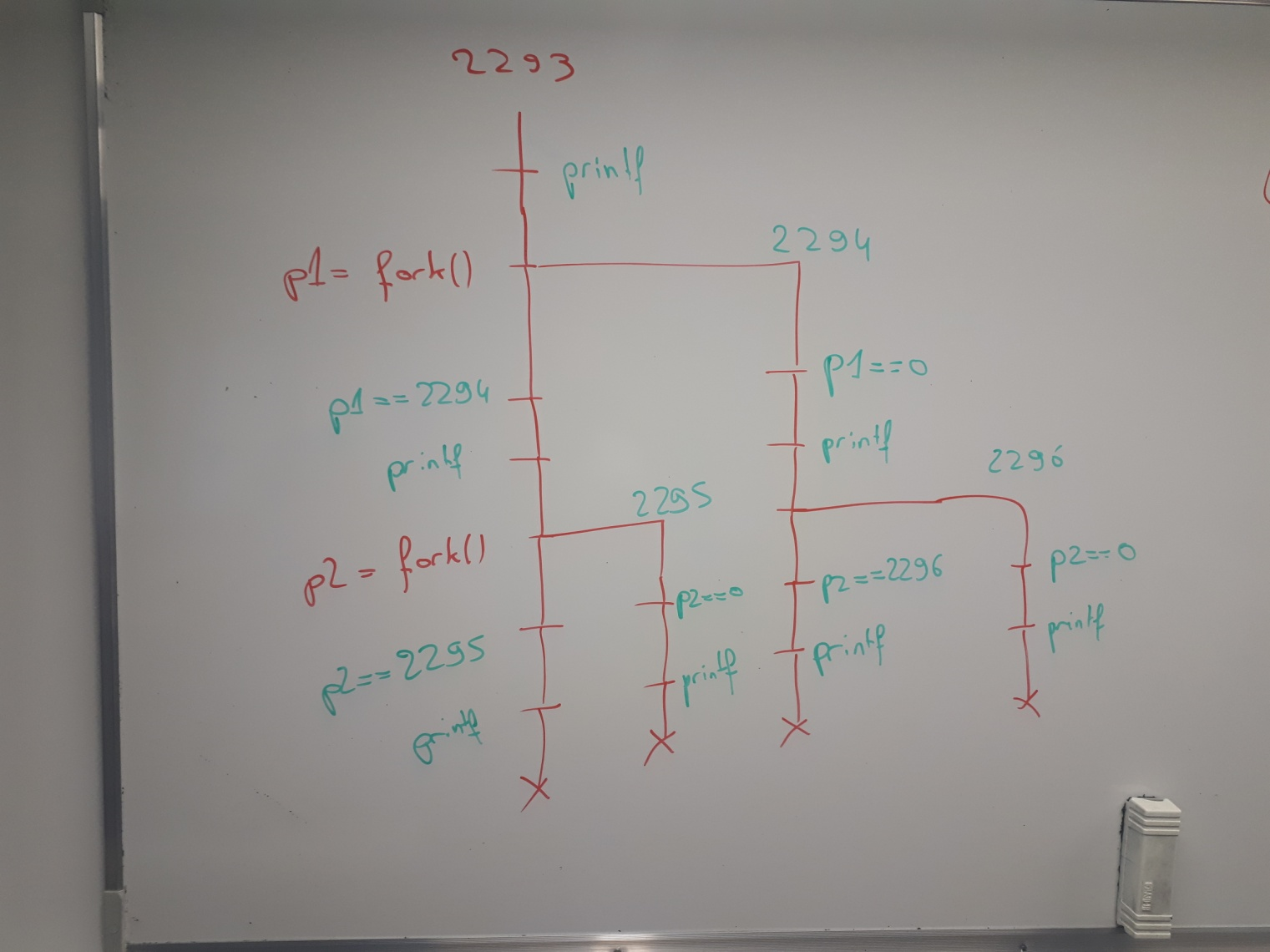
2294 2293 print 2 0

2293 2 print 3 2295

2295 2293 print 3 0

2294 2293 print 3 2296

2296 2294 print 3 0

 Schéma fork()

Analyse :

On peut voir sur celui-ci qu’une fois que le fork() a été appelé une 1ère fois, si nous refaisons un fork(), le fils le fait également. Avec ce fonctionnement cela ne nous fait donc pas 1 processus père et deux fils mais 4 branches (voir ci-dessus).

#### Notes :

* Le schéma précédent montre la création de fils et les valeurs de retour des fork()
* Il ne rend pas compte du PPID, qui peut varier selon les exécutions car l’ordre de terminaison des processus n’est pas déterministe
* L’ordre des printf n’est pas non plus déterministe
* Les numéros des PID sont susceptibles de varier à chaque exécution