

## CI 5 – Processus



# Présentation du cours

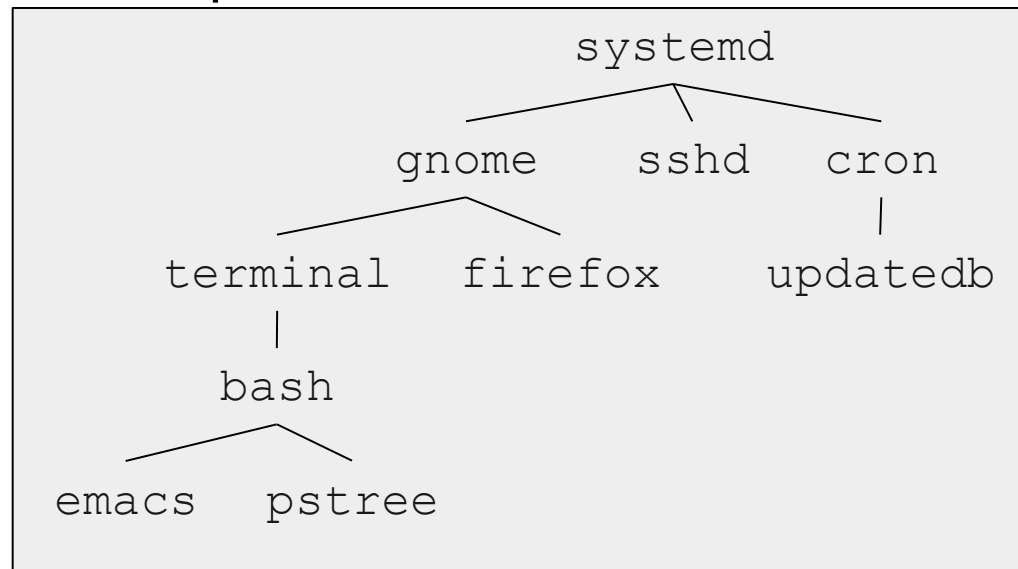
- Contexte :
  - Des dizaines de programmes s'exécutent simultanément sur une machine
- Objectifs :
  - Savoir observer les processus s'exécutant sur une machine
  - Manipuler un processus en cours d'exécution
  - Comprendre comment sont ordonnancés les processus
- Notions clés :
  - Arborescence de processus, états d'un processus, ordonnancement

# Notion de processus

- Processus = programme en cours d'exécution
  - Un espace mémoire
  - Un contexte d'exécution (fichiers ouverts, etc.)
- Caractéristiques statiques
  - **PID** : *Process Identifier*
    - Identifier le processus
  - **PPID** : *Parent Processus Identifier*
    - Hériter de certaines caractéristiques
  - Propriétaire
    - Droits d'accès aux ressources (fichiers, etc.)
- Caractéristiques dynamiques
  - Priorité, environnement d'exécution, etc.
  - Quantité de ressources consommées (temps CPU, etc.)

# Arborescence de processus

- Chaque processus possède un processus parent
  - Sauf le premier processus (`systemd / init`, `PID=1`)
- ➔ arborescence de processus
  - Processus utilisateurs (créés par le shell)
  - *Daemons* : processus qui assurent un service



# Observer les processus

- **ps** : affiche les processus s'exécutant à un instant donné

```
$ ps -l
F S      UID      PID      PPID      C  PRI   NI  ADDR  SZ  WCHAN    TTY          TIME CMD
0 S      1000    22995    1403      0   80    0   -    6285  -          pts/1        00:00:00 bash
0 S      1000    29526    22995      0   80    0   -   128631  -          pts/1        00:00:05 emacs
0 S      1000    29826    22995      0   80    0   -    51571  -          pts/1        00:00:00 oosplash
0 S      1000    29843    29826      1   80    0   -   275029  -          pts/1        00:00:48 soffice.bin
0 R      1000    30323    22995      0   80    0   -     2790  -          pts/1        00:00:00 ps
```

# Observer les processus (suite)

- **ps tree** : affiche l'arborescence des processus

```
$ ps tree -pA
systemd(1) +-ModemManager(535) +-{gdbus}(675)
            |                  +-{gmain}(580)
            | -NetworkManager(552) +-dhclient(27331)
            |                       | -{NetworkManager}(673)
            |                       | -{gdbus}(756)
            |                       +-{gmain}(733)
            | -acpid(692)
            | -konsole(1403) +-bash(22995) +-emacs(29526) +-{dconf worker}(29529)
                                |           |           | -{gdbus}(29528)
                                |           |           +-{gmain}(29527)
                                |           +-pstree(30412)
                                +-{QProcessManager}(1411)
```

# Observer les processus (suite)

- **top** : affiche dynamiquement des processus

```
$ top
top - 15:52:18 up 5 days,  2:04,  3 users,  load average: 0,19, 0,12, 0,13
Tasks: 176 total,  1 running, 175 sleeping,  0 stopped,  0 zombie
%Cpu(s):  6,0 us,  1,3 sy,  0,1 ni, 92,5 id,  0,1 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:  8099392 total, 5840956 used, 2258436 free,  494524 buffers
KiB Swap: 10157052 total,  0 used, 10157052 free. 3114404 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM     TIME+ COMMAND
  866 root        20   0  731892 377196 346672 S   6,4   4,7   21:01.97 Xorg
 1375 trahay      9 -11  651480  11108   8052 S   6,4   0,1   23:23.48 pulseaudio
    1 root        20   0  176840    5420   3144 S   0,0   0,1    0:02.57 systemd
    2 root        20   0      0      0      0 S   0,0   0,0    0:00.01 kthreadd
    3 root        20   0      0      0      0 S   0,0   0,0    0:04.34 ksoftirqd/0
    5 root         0 -20      0      0      0 S   0,0   0,0    0:00.00 kworker/0:0H
    7 root        20   0      0      0      0 S   0,0   0,0    0:30.37 rcu_sched
```

# Détail d'un processus

## ■ `/proc/<pid>/` contient :

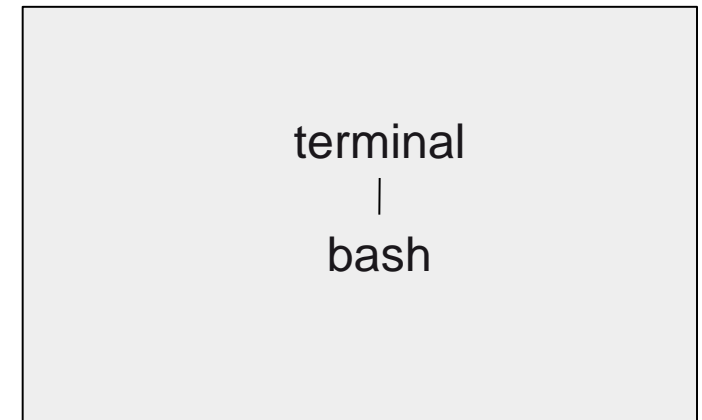
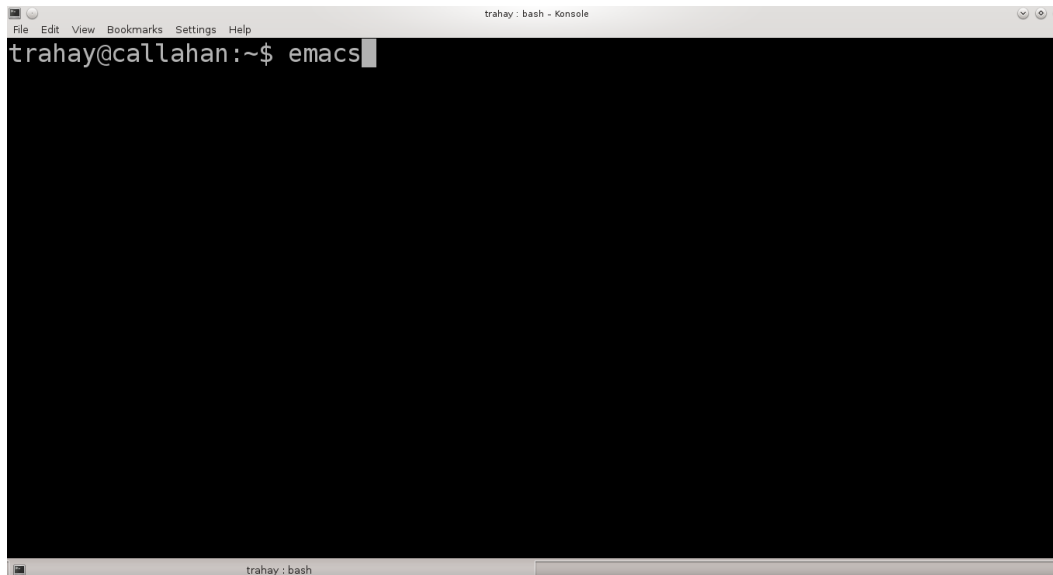
- `cmdline` : texte de la ligne de commande ayant lancé le processus
- `exe` : lien vers le fichier exécutable du programme
- `environ` : contenu de l'environnement
- `fd` : liens vers les fichiers ouverts

```
$ ls /proc/29526
attr          coredump_filter  gid_map          mountinfo        oom_score        sessionid        task
autogroup     cpuset           io               mounts           oom_score_adj    smaps            timers
auxv          cwd              limits           mountstats       pagemap          stack            uid_map
cgroup        environ          loginuid         net              personality      stat             wchan
clear_refs    exe              map_files        ns               projid_map       statm
cmdline       fd               maps             numa_maps        root             status
comm          fdinfo           mem              oom_adj          sched             syscall
```



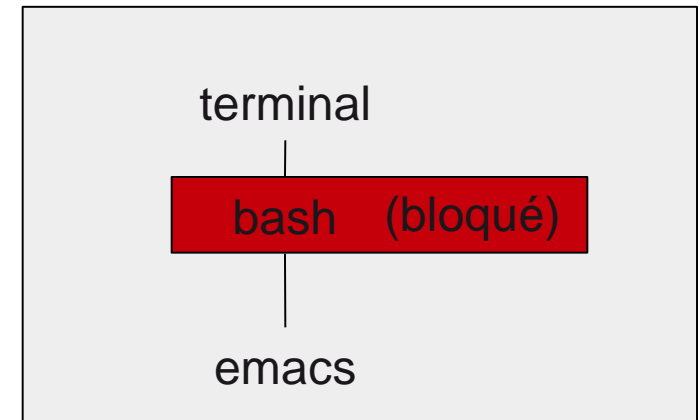
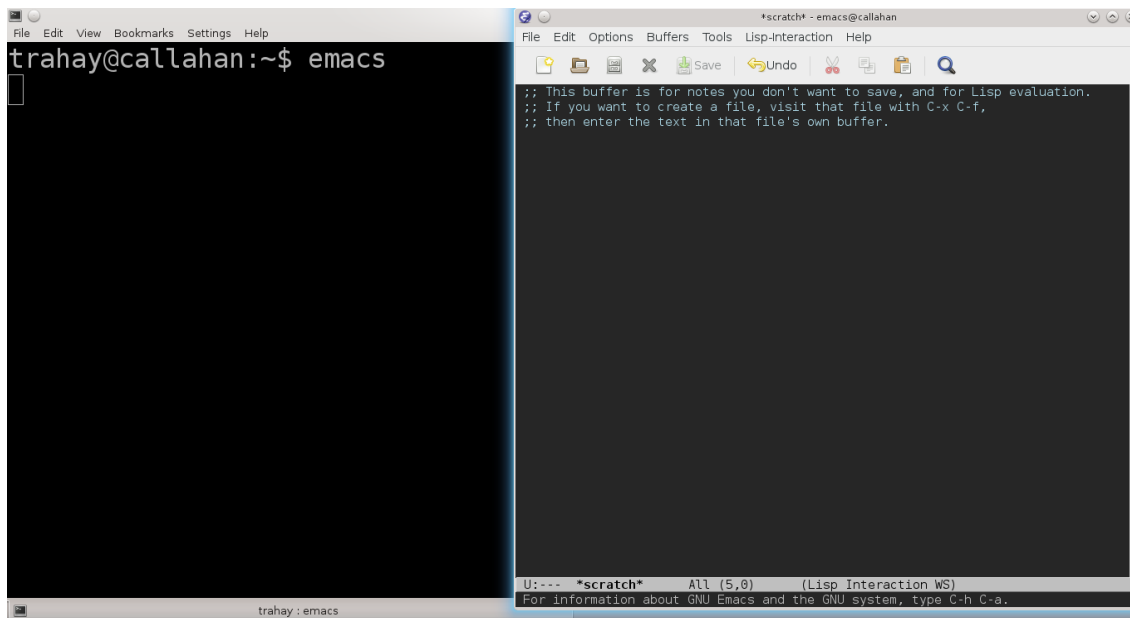
# Processus en avant-plan

- Par défaut, une commande s'exécute en avant-plan (en anglais, *foreground*)
  - Le shell crée un processus enfant et attend qu'il termine
  - Le processus enfant exécute le programme



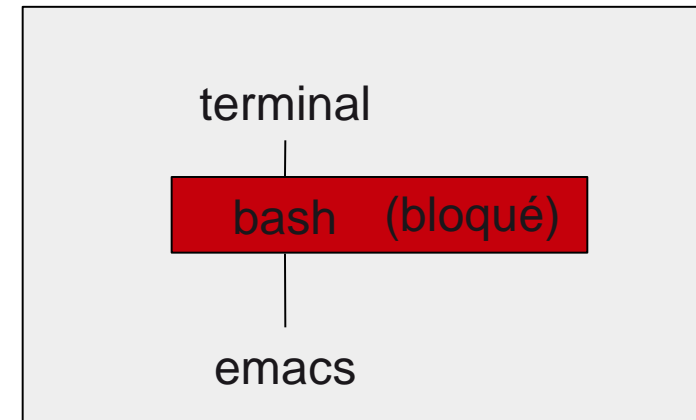
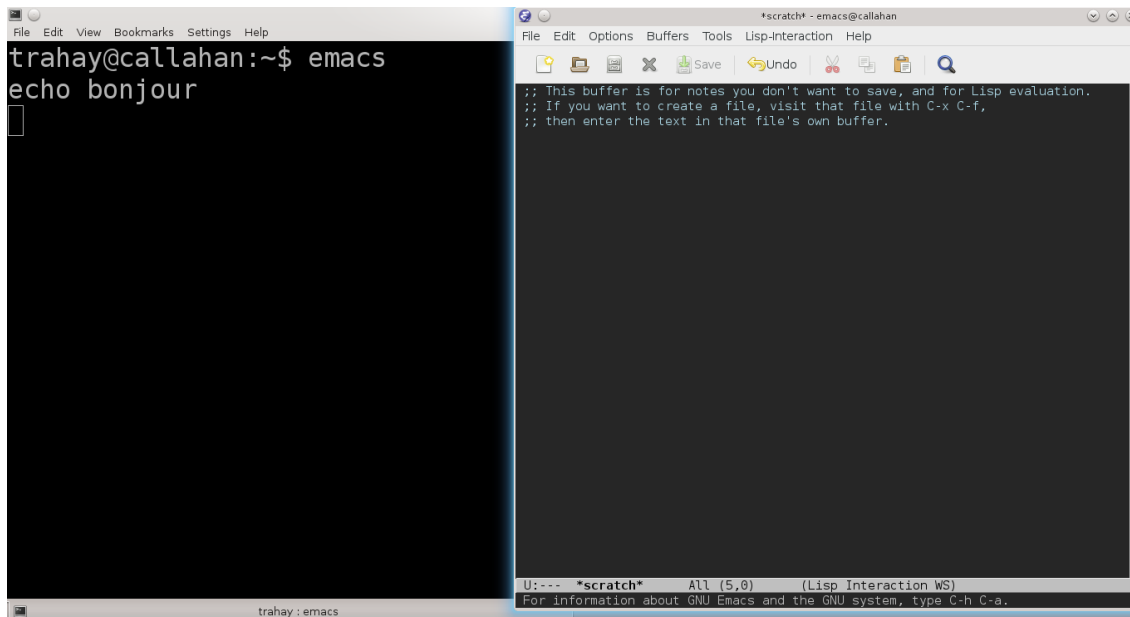
# Processus en avant-plan

- Par défaut, une commande s'exécute en avant-plan (en anglais, *foreground*)
  - Le shell crée un processus enfant et attend qu'il termine
  - Le processus enfant exécute le programme



# Processus en avant-plan

- Par défaut, une commande s'exécute en avant-plan (en anglais, *foreground*)
  - Le shell est bloqué tant que le processus fils s'exécute

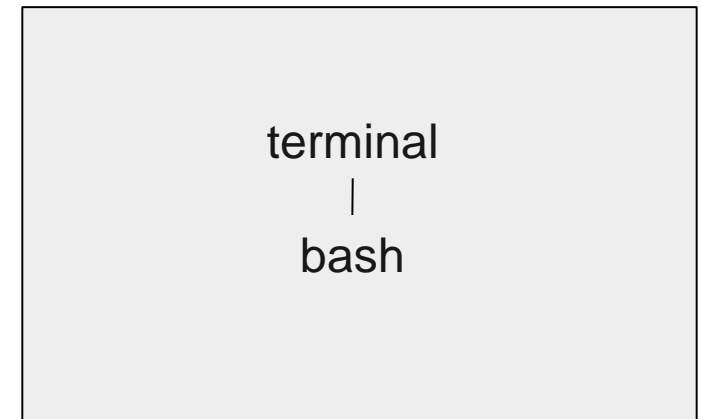


# Processus en avant-plan

- Par défaut, une commande s'exécute en avant-plan (en anglais, *foreground*)
  - Quand le processus fils se termine, il débloquent le shell

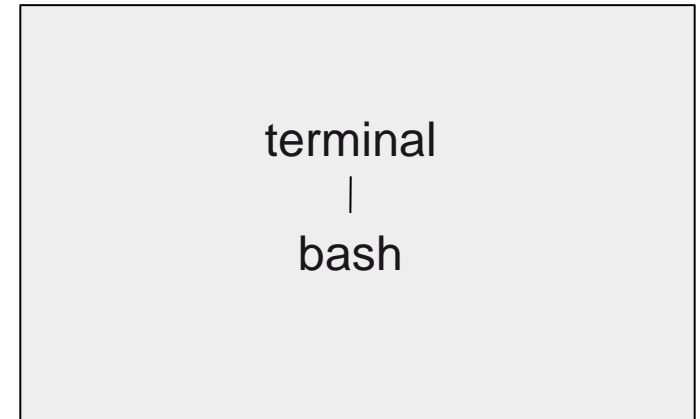
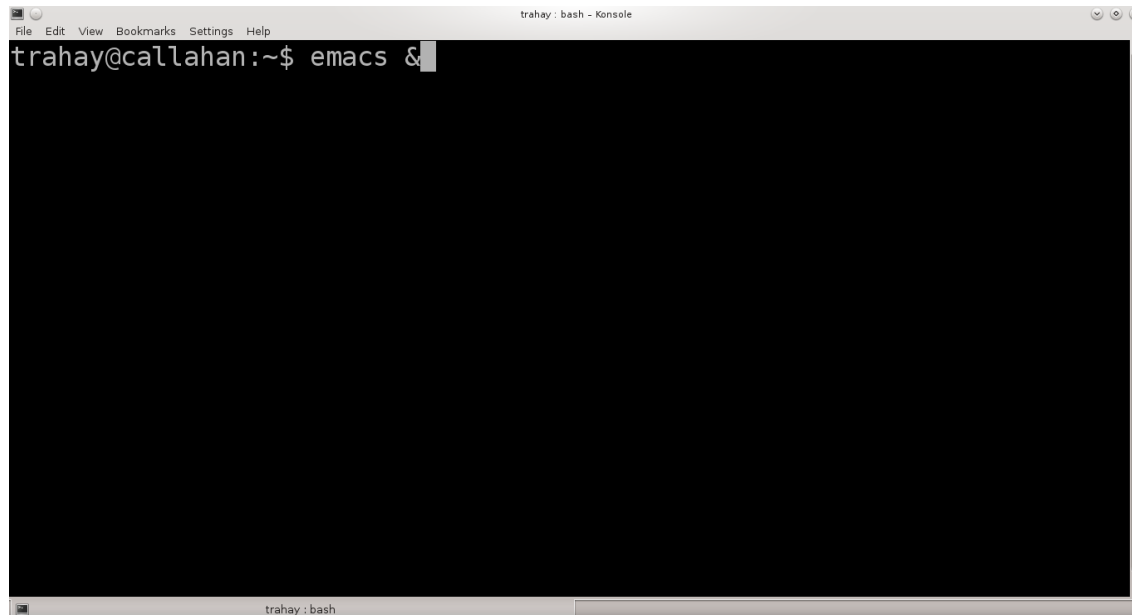


```
trahay@callahan:~$ emacs
echo bonjour
trahay@callahan:~$ echo bonjour
bonjour
trahay@callahan:~$
```



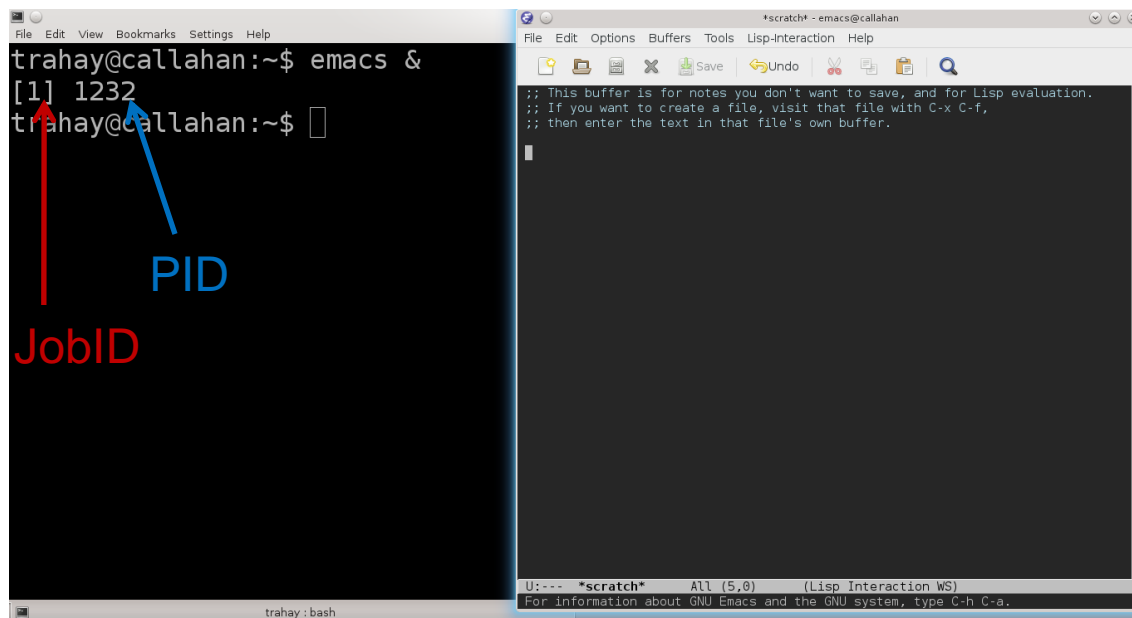
# Processus en arrière-plan

- Pour exécuter une commande arrière-plan (en anglais, *background*) :
  - Terminer la commande par &



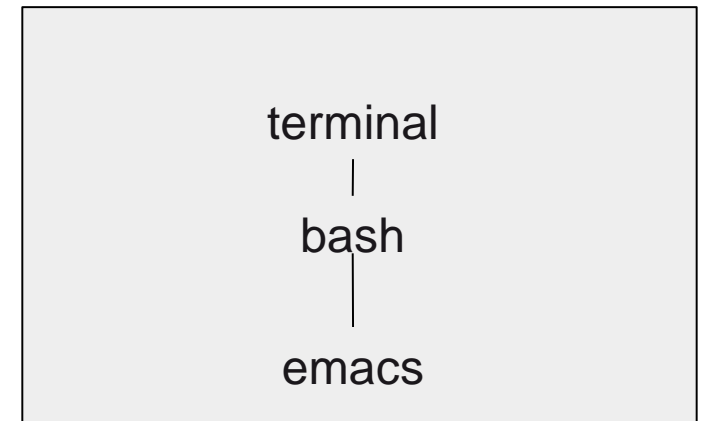
# Processus en arrière-plan

- Commande arrière-plan (en anglais, *background*) :
  - Le shell crée un enfant et n'attend pas qu'il se termine
  - Le shell affiche le **numéro de job (JobID)** et le **PID** du fils
  - Le processus enfant exécute le programme



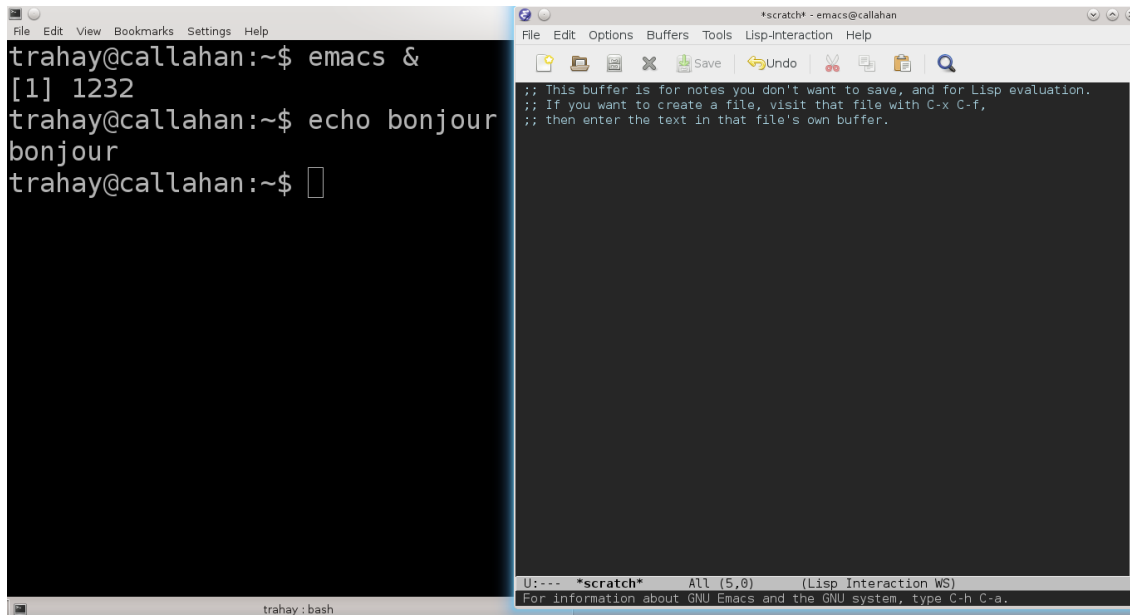
```
trahay@callahan:~$ emacs &
[1] 1232
trahay@callahan:~$
```

The screenshot shows a terminal window with the command `emacs &` entered. The prompt changes to `[1] 1232`, indicating the job ID and PID. A red arrow points to `[1]` with the label **JobID**, and a blue arrow points to `1232` with the label **PID. To the right, an Emacs editor window is open, showing a buffer for notes.**



# Processus en arrière-plan

- Commande arrière-plan (en anglais, *background*) :
  - Le shell et le processus fils s'exécutent en parallèle
  - Le shell peut exécuter d'autres commandes

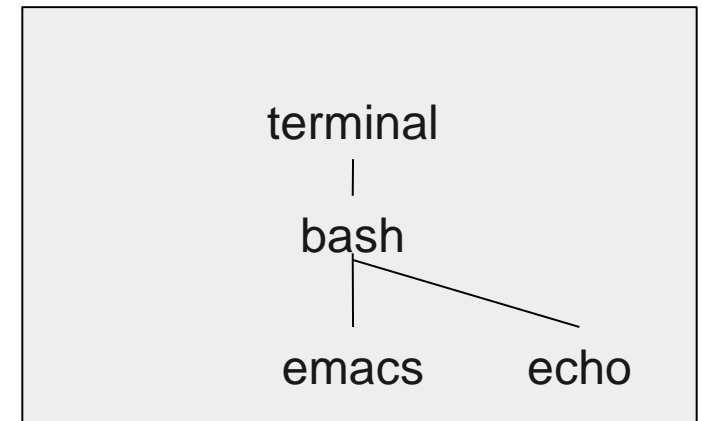


The image shows two overlapping windows. The left window is a terminal with the following text:

```
trahay@callahan:~$ emacs &  
[1] 1232  
trahay@callahan:~$ echo bonjour  
bonjour  
trahay@callahan:~$
```

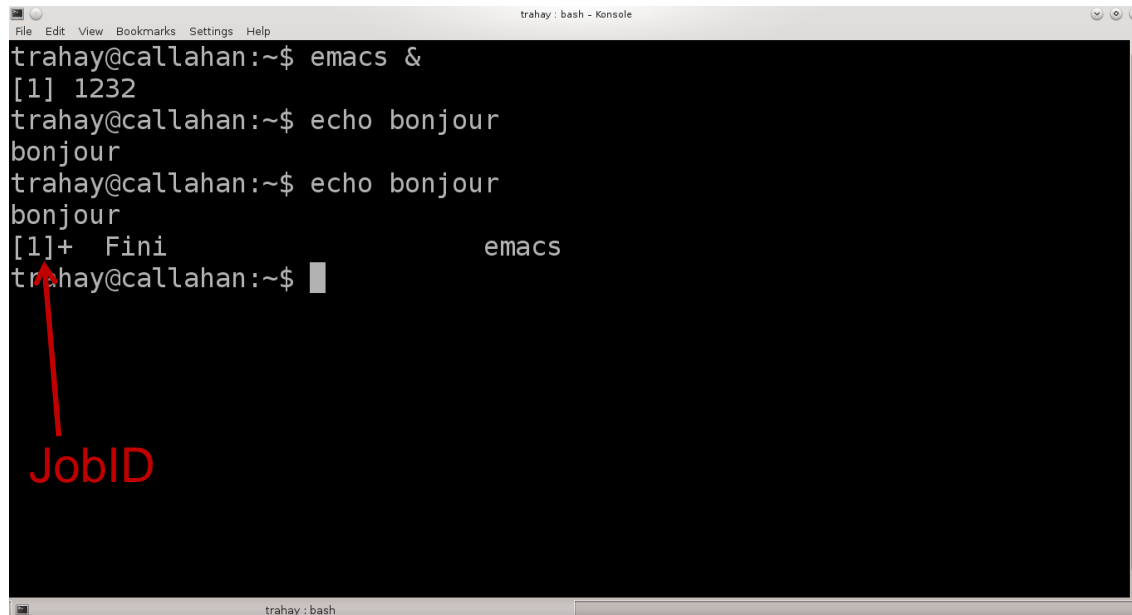
The right window is an Emacs editor titled '\*scratch\* - emacs@callahan'. It contains the following text:

```
;; This buffer is for notes you don't want to save, and for Lisp evaluation.  
;; If you want to create a file, visit that file with C-x C-f,  
;; then enter the text in that file's own buffer.
```



# Processus en arrière-plan

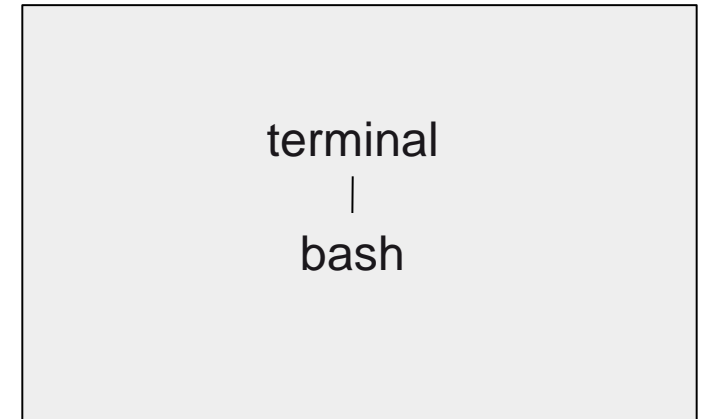
- Commande arrière-plan (*background*) :
  - Quand le fils se termine, il en informe le shell



A screenshot of a terminal window titled "trahay : bash - Konsole". The terminal shows the following commands and output:

```
trahay@callahan:~$ emacs &
[1] 1232
trahay@callahan:~$ echo bonjour
bonjour
trahay@callahan:~$ echo bonjour
bonjour
[1]+  Fini                  emacs
trahay@callahan:~$
```

A red arrow points from the text "JobID" to the "[1]" in the first line of output. The word "emacs" is also visible in the output line "[1]+ Fini emacs".





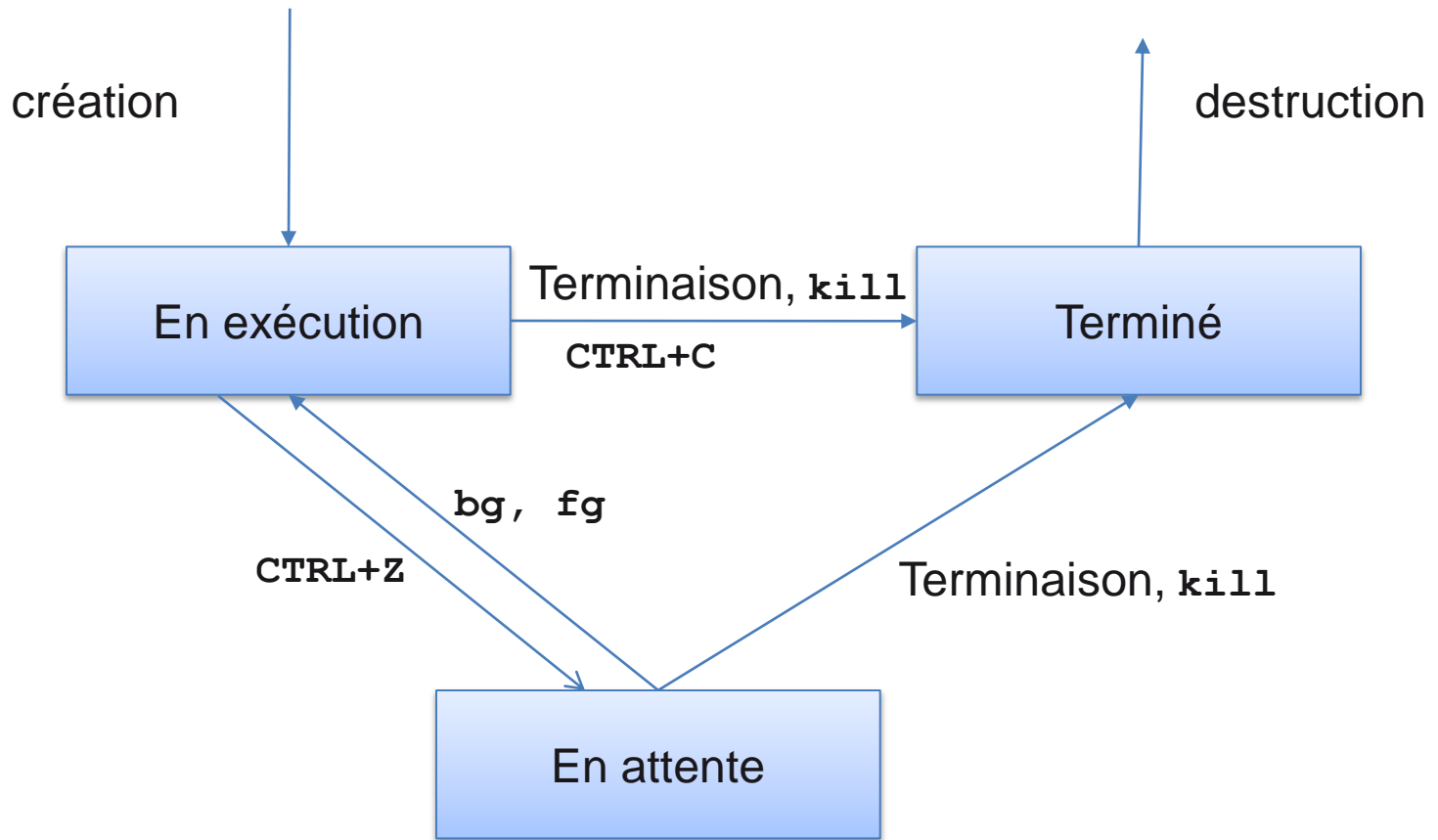
# Suspendre un processus

- Suspendre un processus en avant-plan : **Ctrl+Z**
  - Le processus est placé « en attente »
- Reprendre un processus en attente
  - Pour le mettre en avant-plan : **fg** (*foreground*)
  - Pour le mettre en arrière-plan : **bg** (*background*)

# Suppression d'un processus

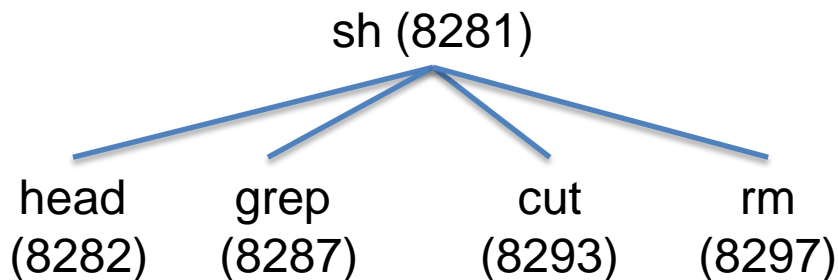
- Un processus se termine s'il atteint sa dernière instruction
- Arrêter un processus en avant-plan : **Ctrl+C**
- Arrêter un processus en arrière-plan : **kill %<JobID>**
- Arrêter un processus quelconque : **kill <PID>**
  - Envoie un signal (**SIGTERM**) au processus **<PID>**
  - Envoi du signal de destruction (**SIGKILL**) : **kill -9 <PID>**
- Arrêter un ensemble de processus : **killall <programme>**
  - Envoie un signal (**SIGTERM**) à tous les processus **<programme>**

# États d'un processus



# Shell et processus

- Commandes dans un script shell → nouveaux processus
- Variables utilisable depuis un script shell
  - \$\$ : PID du shell exécutant le script
  - \$PID : PPID du shell exécutant le script



```
#!/bin/sh
```

```
head -n 30 itineraire > debut_iti  
grep plage debut_iti > baignade  
cut -d' ' -f3 baignade > tresor  
rm baignade
```

**vacances.sh**

# Shell et processus (suite)

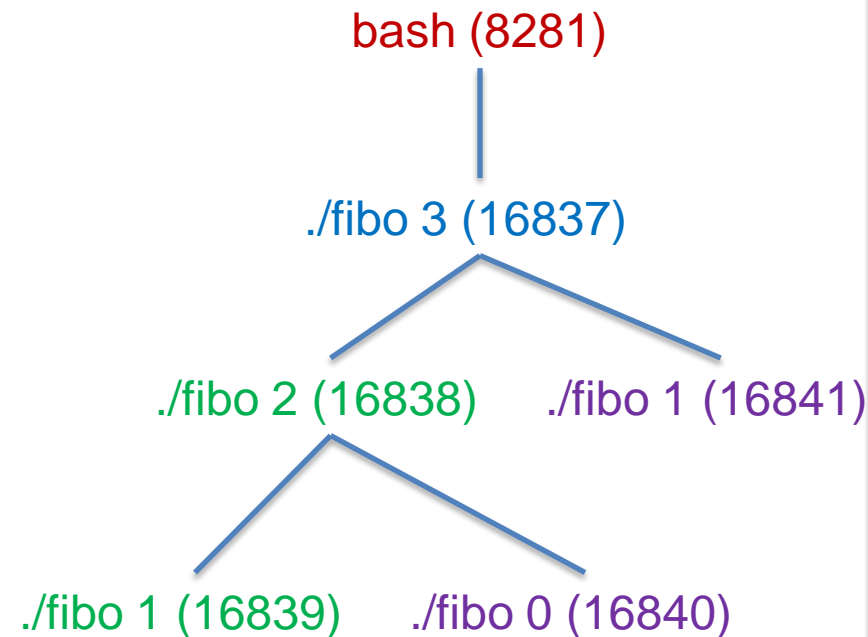
□ Lancement d'un script shell dans un processus enfant:

- `$ bash ./fibonacci.sh`
- `$ ./fibonacci.sh`

(si `fibonacci.sh` possède l'attribut `x`)

```
#!/bin/sh
if [ $1 -eq 0 ] || [ $1 -eq 1 ];
then
    echo 1
else
    n=$1
    fib1=`bash ./fibonacci.sh $((n-1))`
    fib2=`bash ./fibonacci.sh $((n-2))`
    echo $(( fib1 + fib2 ))
fi
```

`fibonacci.sh`



# Shell et processus

- Lancement d'un script shell dans le processus shell courant:
  - `$ . ./fibonacci.sh` (en utilisant la fonction « `.` »)
  - `$ source ./fibonacci.sh`

sh (8281)

```
#!/bin/sh
if [ $1 -eq 0 ] || [ $1 -eq 1 ];
then
    echo 1
else
    n=$1
    fib1=`. ./fibonacci.sh $((n-1))`
    fib2=`. ./fibonacci.sh $((n-2))`
    echo $(( fib1 + fib2 ))
fi
```

**fibonacci.sh**

# Gestion de processus

- Comment exécuter plusieurs processus simultanément ?

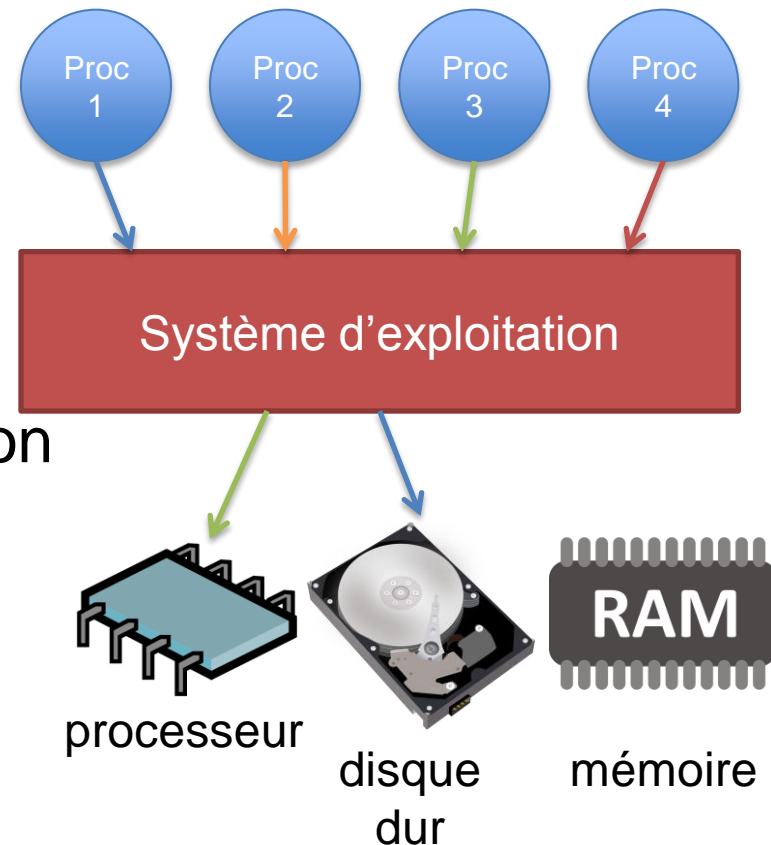
# Partage de ressources

## ■ Ressources partagées par les processus

- CPU (cœur d'un processeur)
- Mémoire
- Entrées-sorties

## ■ Gestion par le Système d'Exploitation

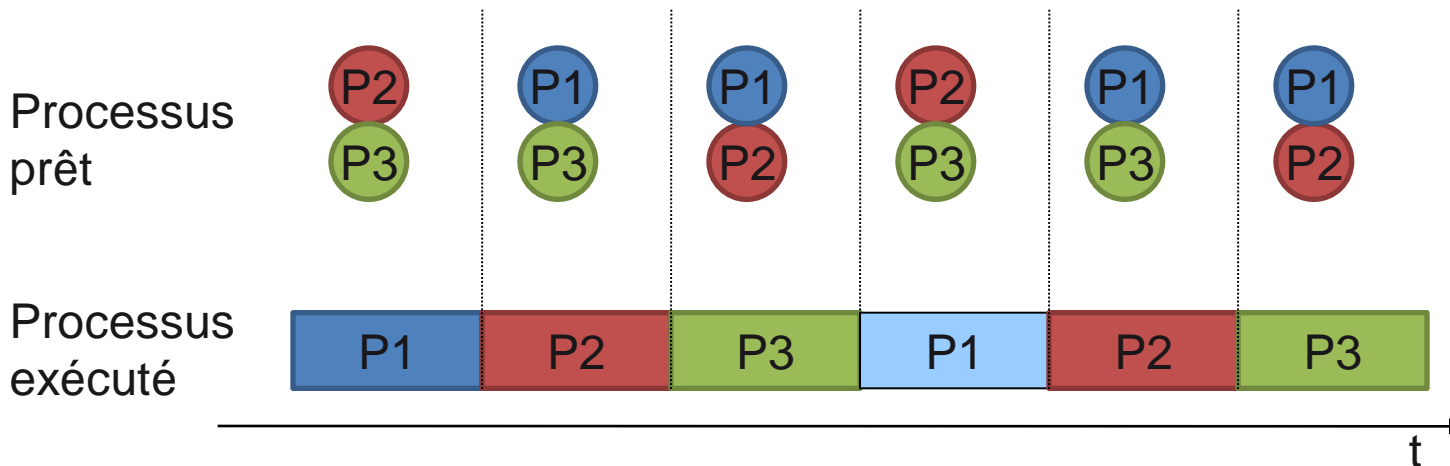
- Exclusion mutuelle
- Contrôle de l'accès au matériel
  - Droits d'accès
  - Non-dépassement des limites





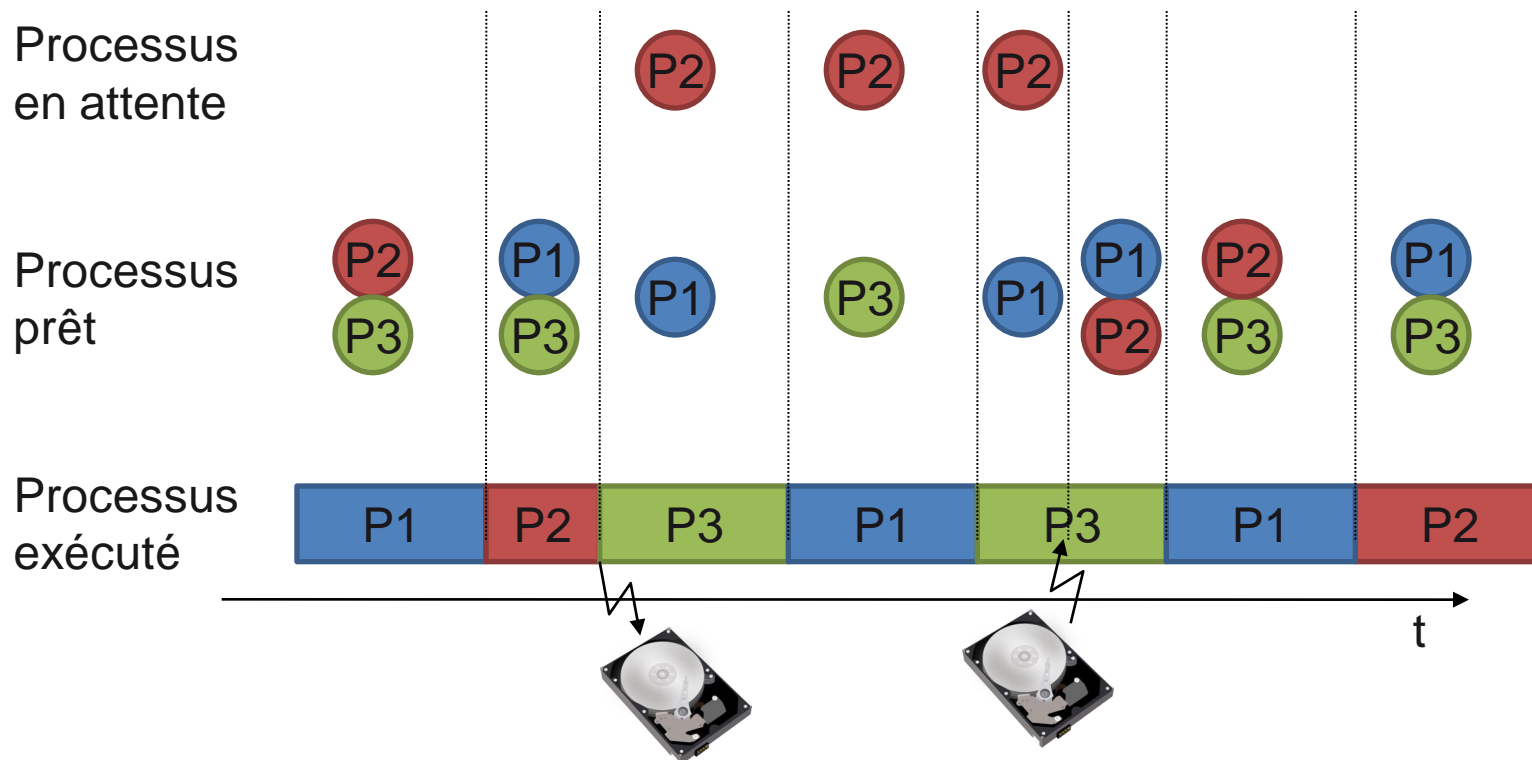
# Partage du CPU

- À un instant donné, le CPU n'exécute qu'un processus
  - Les autres processus attendent
- L'ordonnanceur partage le CPU par « quantum de temps » (en anglais, *timeslice*)
  - À la fin du *timeslice*, l'ordonnanceur préempte le processus s'exécutant et choisit un autre processus



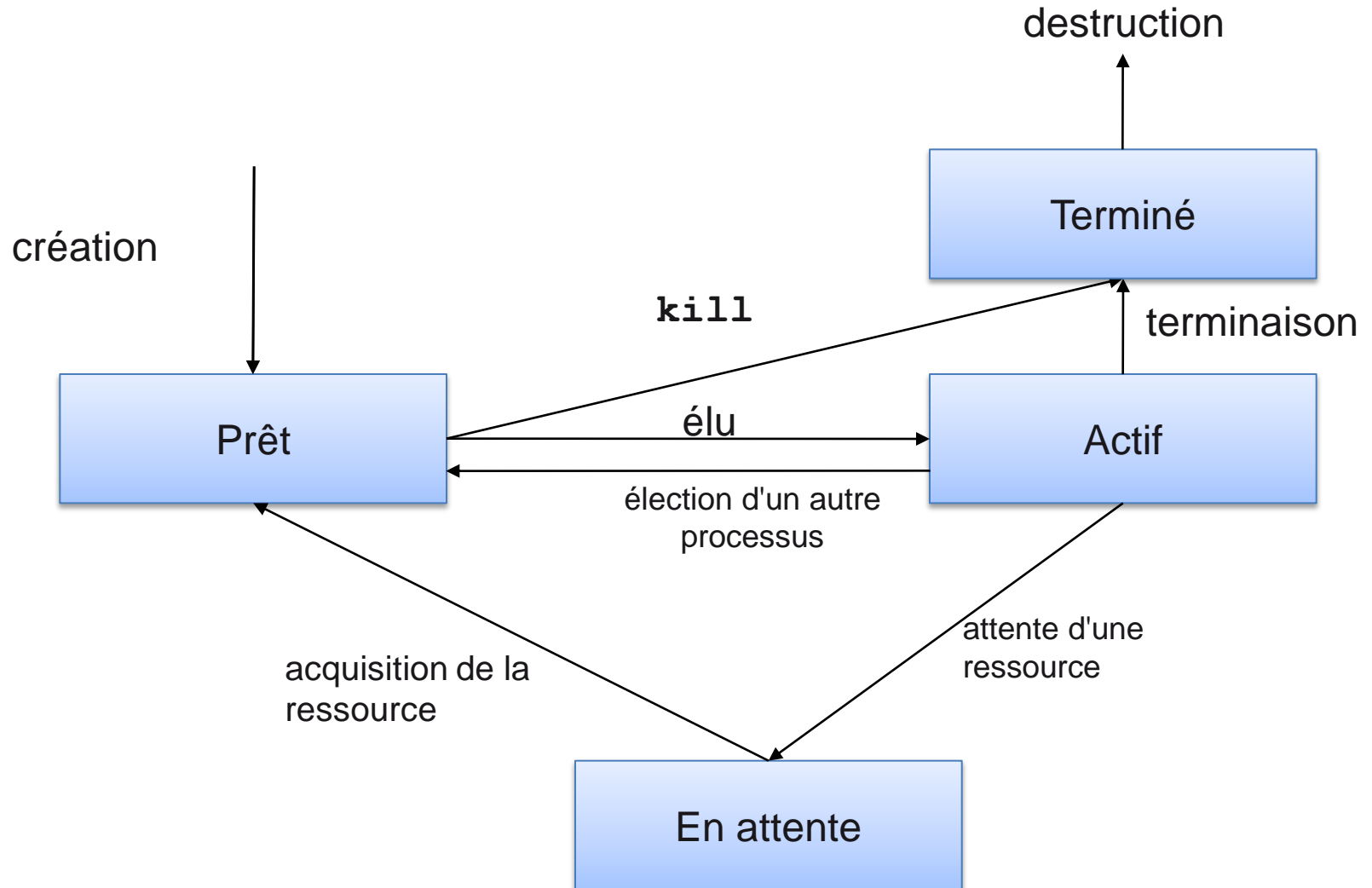
# Partage du CPU entrées/sorties

- Entrées/sorties → attente d'une ressource (disque, carte réseau, écran, etc.)
- Libération du CPU en attendant la ressource



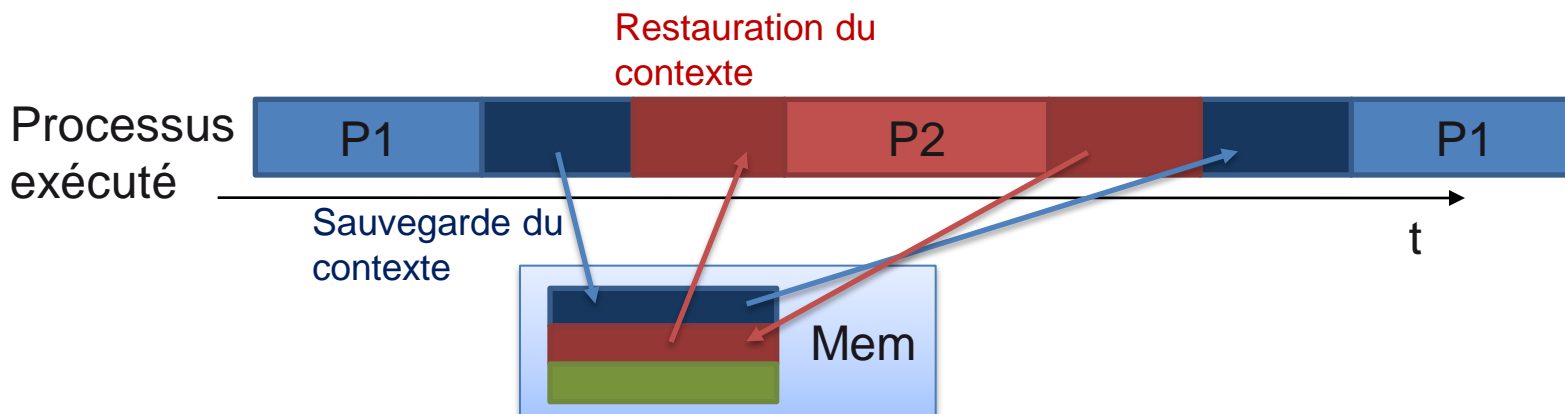
# États d'un processus

Le point de vue du système d'exploitation



# Commutation de processus

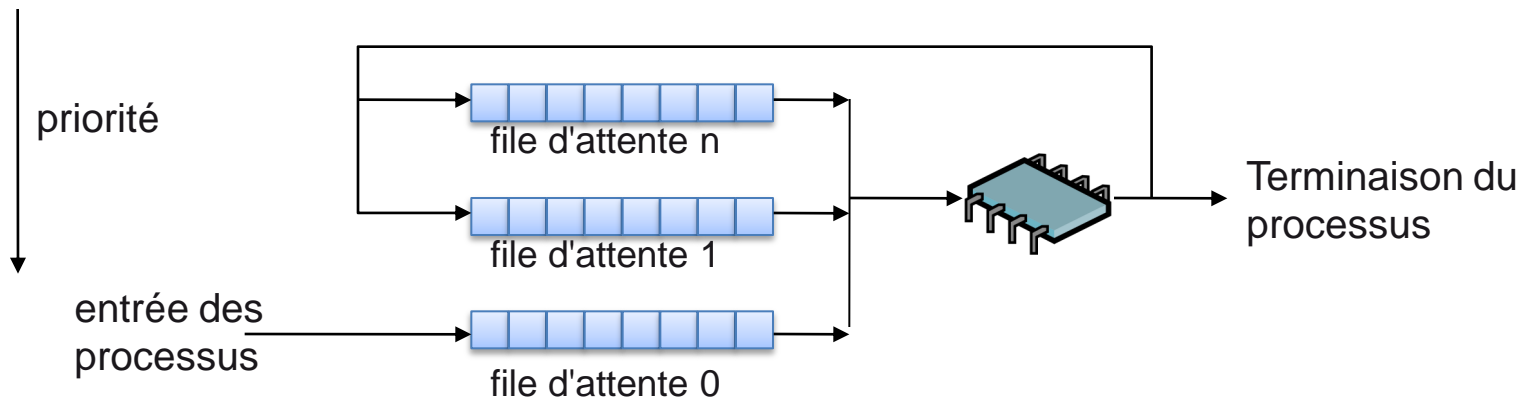
- La commutation a lieu lors de l'élection d'un processus:
  - Sauvegarde du contexte du processus évincé
  - Chargement du contexte du processus élu
- Contexte : ensemble des infos associées au processus
  - Valeur des registres
  - Informations mémoire (emplacement, etc.)



# Ordonnancement de processus

## Exemple d'algorithme d'ordonnancement à priorité

- Une file d'attente des processus prêts par niveau de priorité
  - L'ordonnanceur choisit plus souvent les processus de forte priorité
  - Ajustement de la priorité d'un processus au court de son exécution
- Exemple d'algorithme d'ordonnancement
    - Choisir  $n$  processus de `priorité 0`
    - Choisir  $n-1$  processus de `priorité 1`
    - [...]
    - Choisir 1 processus de `priorité n`
    - Si un processus consomme tout son `timeslice` : `priorité--`
    - Si un processus rend la main : `priorité++`



# Changer la priorité d'un processus

- Possibilité de changer manuellement la priorité d'un processus
  - Exemple: baisser la priorité d'un programme qui indexe le contenu d'un disque dur
- Lancer un programme avec une certaine priorité
  - `$ nice -n <priorité> <commande>`
- Changer la priorité d'un processus déjà lancé
  - `$ renice -n <priorité> <PID>`

# Introduction à la concurrence

- Accès concurrent à une ressource gérée par l'OS
  - Disque dur, imprimante, écran, ...
- L'OS assure l'*exclusion mutuelle*
  - À tout moment, seul un processus manipule la ressource

```
$ do_ping & do_pong
```

```
ping  
pong  
ping  
pong  
ping  
pong  
ping  
pong  
pong  
ping  
ping  
ping  
ping  
pong
```

```
#!/bin/bash  
while [ true ]; do  
    echo ping  
done
```

do\_ping

```
#!/bin/bash  
while [ true ]; do  
    echo pong  
done
```

do\_pong



t

# Conclusion

## □ Concepts clés

- Processus
  - Caractéristiques statiques et dynamiques
  - Processus parent, processus enfant
  - Exécution en avant-plan, arrière-plan, suspension/reprise de processus
- Ordonnancement de processus
  - Quantum de temps, préemption
  - changement de contexte

## □ Commandes clés

- `ps`, `pstree`, `top`
- `CTRL+Z`, `fg`, `bg`
- `CTRL+C`, `kill`, `killall`



# En route pour le TP !!