

IF 110 – Les Processus



Laurent Réveillère

Université de Bordeaux

`laurent.reveillere@u-bordeaux.fr`

`http://www.reveillere.fr/`

D'après le cours d'introduction aux systèmes d'exploitation de Télécom SudParis



Présentation du cours

□ Contexte :

- Des dizaines de processus 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 + contexte d'exécution (fichiers ouverts, etc.)
- ❑ Caractéristiques statiques
 - PID : Process Identifier (identifie le processus)
 - PPID : Parent Processus Identifier (identifie le parent)
 - Utilisateur propriétaire
 - Droits d'accès aux ressources (fichiers, etc.)
- ❑ Caractéristiques dynamiques
 - Priorité, environnement d'exécution, etc.
 - Quantité de ressources consommées (temps CPU, etc.)



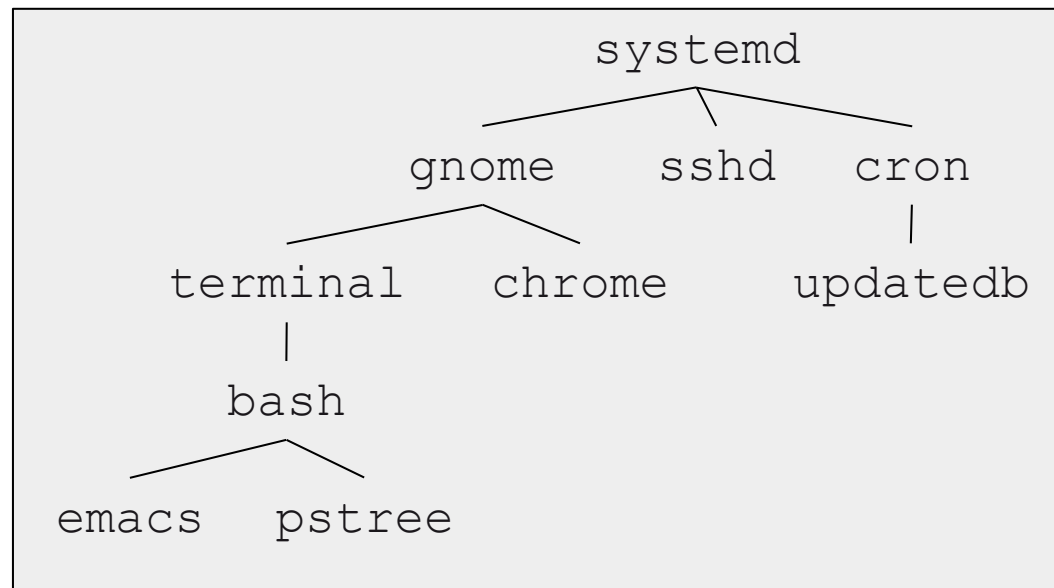
Plan

1. Observer un processus
2. Processus en avant et arrière plan
3. Cycle de vie d'un processus
4. Variables et processus
5. Gestion des processus dans le système d'exploitation



Arborescence de processus

- ❑ Chaque processus possède un processus parent
 - Sauf le premier processus (`systemd` ou `init`, PID=1)
⇒ arborescence de processus
- ❑ Deux types de processus :
 - Processus utilisateurs (attachés à un terminal)
 - Daemons : processus qui assurent un service (détachés de tout terminal)





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
```



```
$ pstree -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
```



Variables relatives aux processus

- ❑ Chaque processus `bash`, y compris les scripts, définissent :
 - `$$` : PID du `bash` courant
 - `$PPID` : PID du parent du `bash` courant

```
$ echo $$  
20690  
$ echo $PPID  
20689  
$
```



Variables relatives aux processus

- ❑ Chaque processus `bash`, y compris les scripts, définissent :
 - `$$` : PID du `bash` courant
 - `$PPID` : PID du parent du `bash` courant

```
$ echo $$
20690
$ echo $PPID
20689
$ ps -p 20689,20690
  PID TTY          TIME CMD
20689 ??            0:11.69 xterm -r
20690 ttys004      0:01.32 bash
$
```



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	



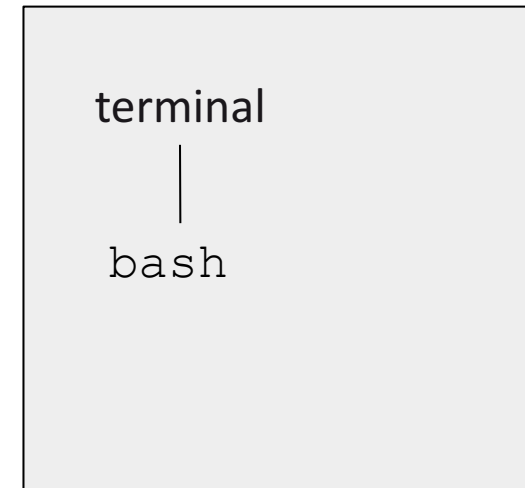
Plan

1. Observer un processus
2. Processus en avant et arrière plan
3. Cycle de vie d'un processus
4. Variables et processus
5. Gestion des processus dans le système d'exploitation



Processus en avant-plan

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

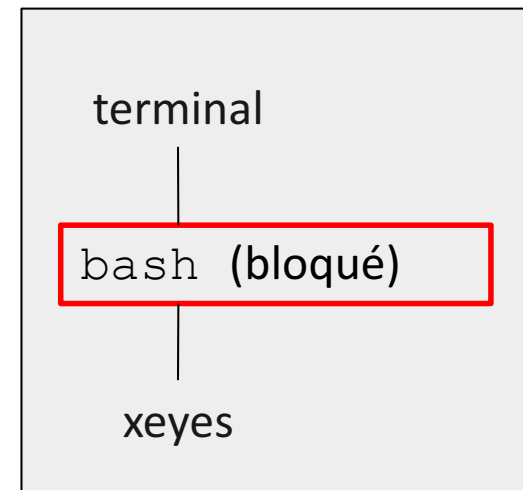
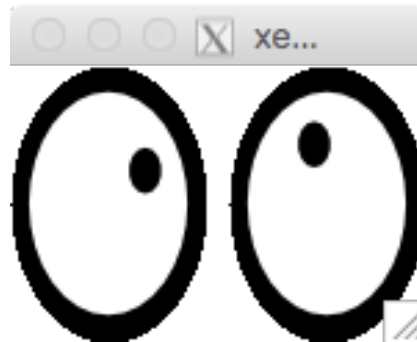




Processus en avant-plan

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

```
$ xeyes
```





Processus en avant-plan

- ❑ Par défaut, une commande s'exécute en avant-plan (en anglais, *foreground*)
 - Quand le processus fils se termine, `bash` reprend son exécution

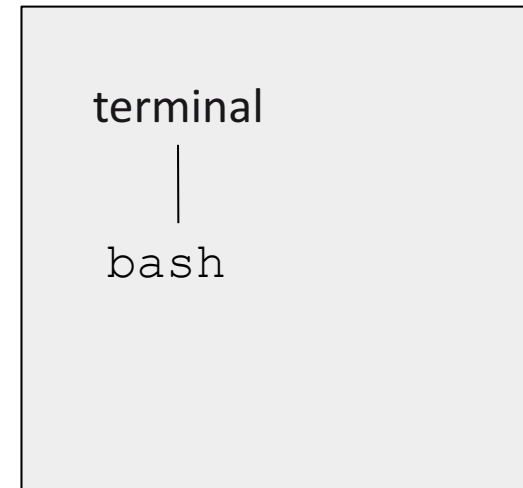
```
$ xeyes  
$
```

```
terminal  
|  
bash
```



Processus en arrière-plan

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



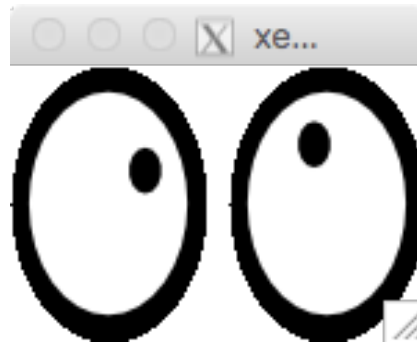


Processus en arrière-plan

❑ Commande en arrière-plan (en anglais, *background*)

- bash crée un enfant et n'attend pas qu'il se termine
- bash affiche le **numéro de job (JobID)** et le **PID** du fils
- Le processus enfant exécute le programme

```
$ xeyes &  
[1] 35794  
$
```



```
terminal  
|  
bash  
|  
xeyes
```



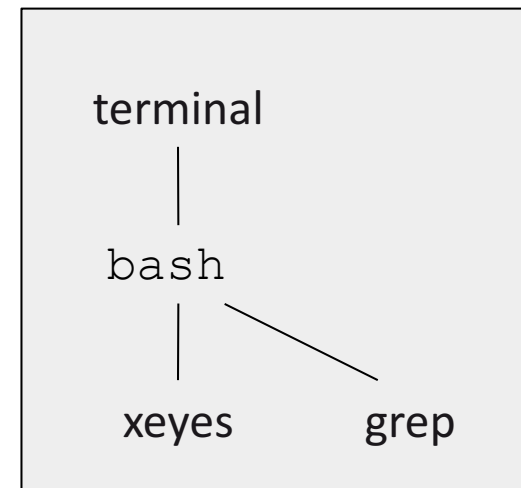
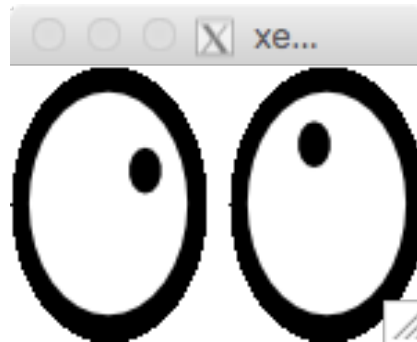
Processus en arrière-plan

❑ Commande en arrière-plan (en anglais, *background*)

:

- bash et le processus fils s'exécutent en parallèle
- bash peut donc exécuter d'autres commandes

```
$ xeyes &  
[1] 35794  
$ grep c bjr.txt  
coucou
```





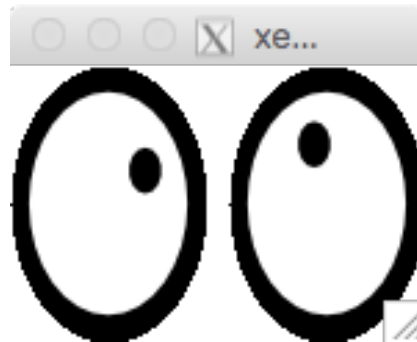
Processus en arrière-plan

❑ Commande en arrière-plan (en anglais, *background*)

:

- bash et le processus fils s'exécutent en parallèle
- bash peut donc exécuter d'autres commandes

```
$ xeyes &  
[1] 35794  
$ grep c bjr.txt  
coucou  
$
```



```
terminal  
|  
bash  
|  
xeyes
```



Processus en arrière-plan

- ❑ Commande en arrière-plan (background) :
 - Quand le fils se termine, le système d'exploitation informe bash

```
$ xeyes &  
[1] 35794  
$ grep c bjr.txt  
coucou  
$  
[1]+  Done    xeyes  
$
```

JobID

terminal

bash



PID du dernier processus lancé

- ❑ Le PID du dernier processus lancé est dans la variable \$!

```
$ xeyes &  
[1] 35794  
$ echo $!  
35794  
$
```



Plan

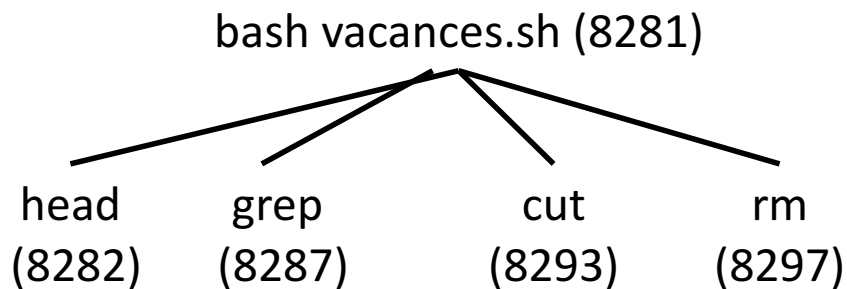
1. Observer un processus
2. Processus en avant et arrière plan
3. Cycle de vie d'un processus
4. Variables et processus
5. Gestion des processus dans le système d'exploitation



Commandes et processus

❑ Chaque commande crée un processus

Sauf pour les commandes internes qui sont directement interprétées par `bash` (`exit`, `source`...)



```
#!/bin/bash
```

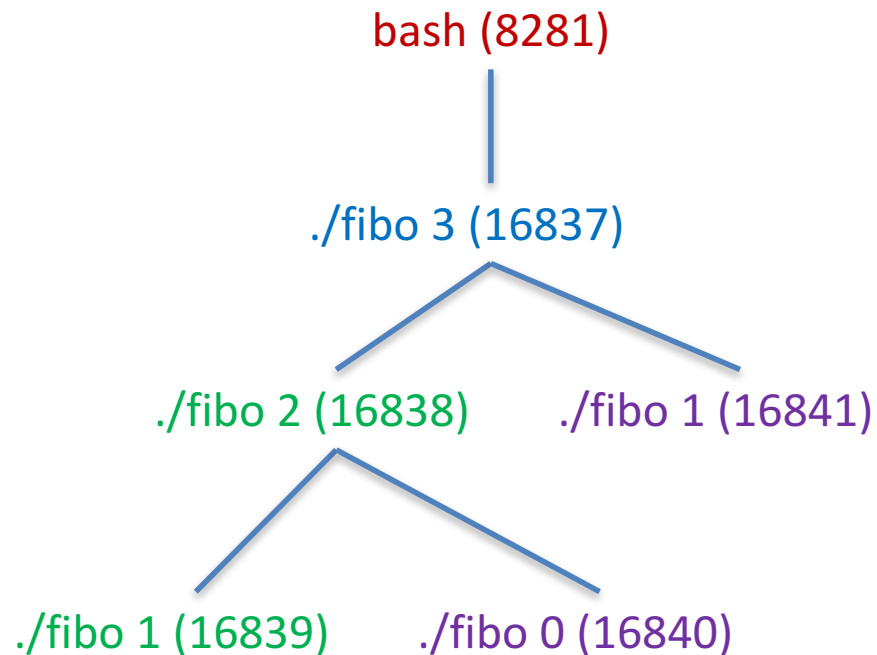
```
head -n 30 itineraire > debut iti  
grep plage debut iti > baignade  
cut -d' ' -f3 baignade > tresor  
rm baignade
```

`vacances.sh`



Scripts et processus

- Par défaut, un script est lancé dans un processus enfant



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

fibo



Scripts et processus

- ❑ On peut aussi charger un script dans le processus
bash courant avec `source ./script.sh`



Suspendre un processus

- ❑ Suspendre un processus en avant-plan :
`control+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*)

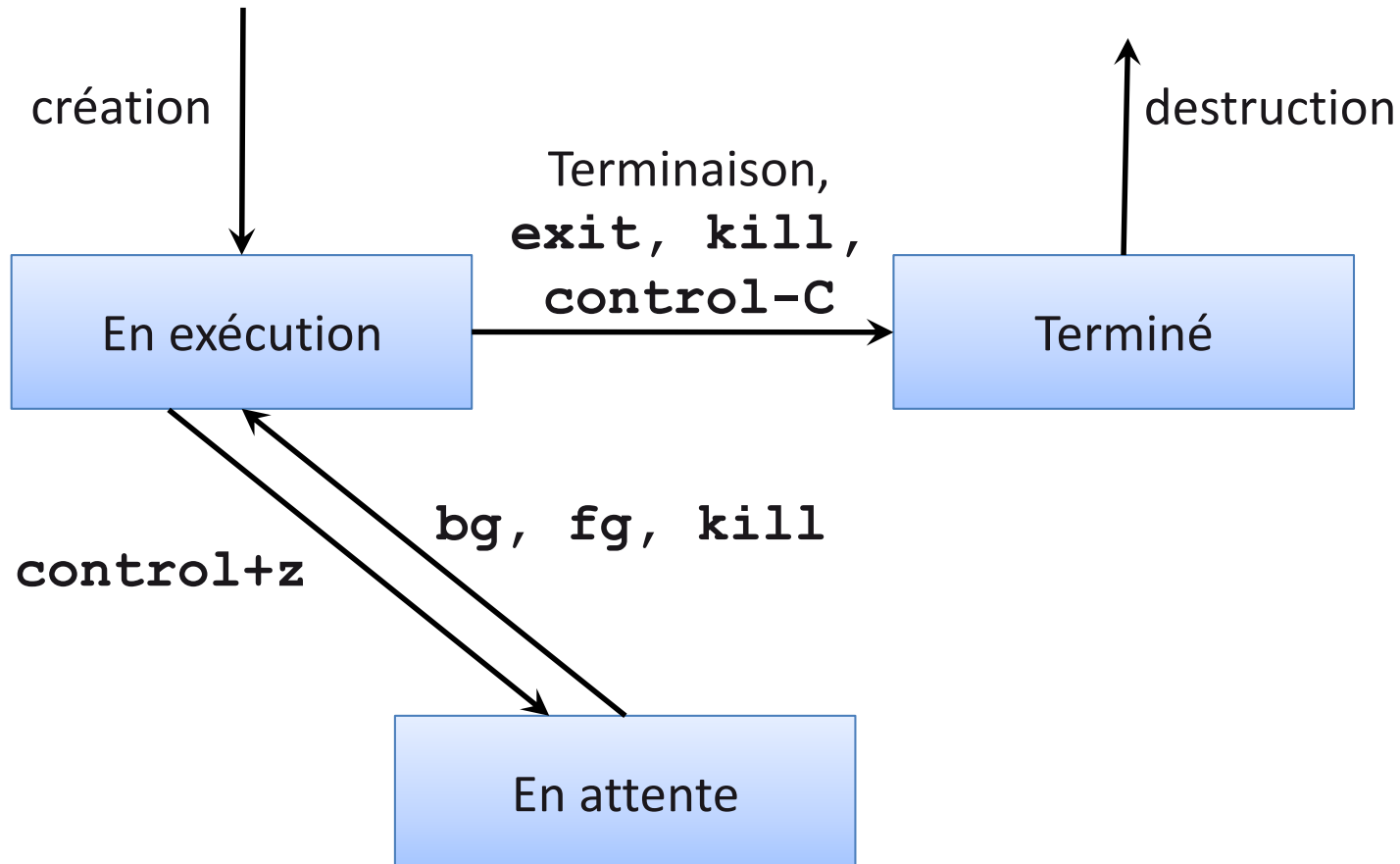


Suppression d'un processus

- ❑ Un processus se termine s'il atteint sa dernière instruction
- ❑ Ou s'il appelle `exit`
- ❑ Ou s'il reçoit un signal (voir cours suivant)
 - `control-c` : tue le processus en avant plan (avec `SIGINT`)
 - `kill` ou `killall` : tue un processus (avec `SIGTERM`)
 - » `kill %JobID` : tue le processus de numéro de job `JobID`
 - » `kill PID` : tue le processus d'identifiant `PID`
 - » `killall prog` : tue tous les processus dont le chemin du programme est `prog`
 - **Remarque** : vous verrez plus tard que les processus peuvent résister à `control-c`, `kill` ou `killall`. Si c'est le cas, ajoutez `-9` (`SIGKILL`) après `kill/killall` pour forcer leur mort



États d'un processus





Attendre la fin d'un processus

□ La commande `wait` permet d'attendre la fin d'un fils

➤ `wait` sans argument : attend la fin de tous les fils

➤ `wait %jobid1 %jobid2... ou wait pid1 pid2... :`
attend la fin des processus passés en argument



Attendre la fin d'un processus



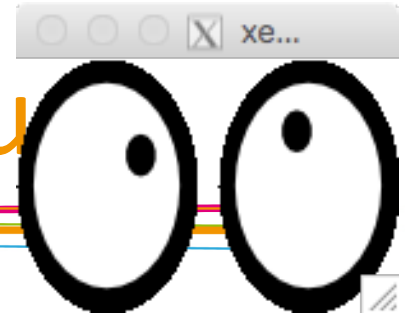
bash _____

temps

_____ Processus en exécution



Attendre la fin d'un processus



```
$ xeyes &  
$
```

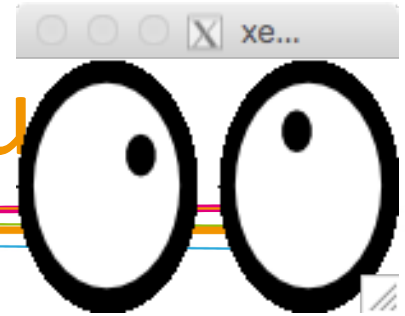


— Processus en exécution

— → Création de processus



Attendre la fin d'un processus



```
$ xeyes &  
$ grep reveillere /etc/passwd
```



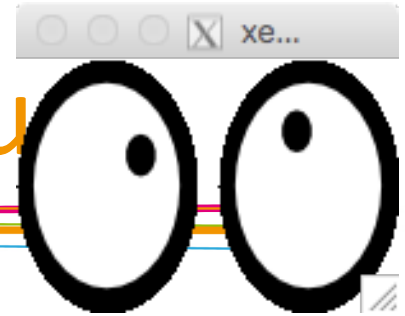
— Processus en exécution

— → Création de processus

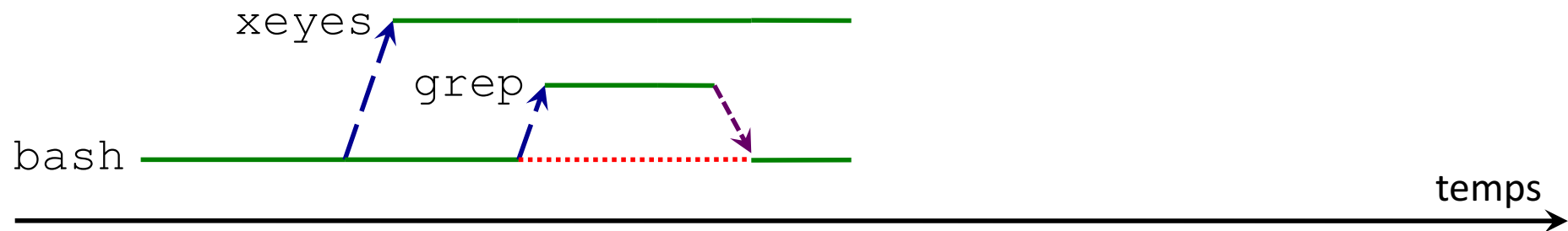
..... Processus en attente



Attendre la fin d'un processus



```
$ xeyes &  
$ grep reveillere /etc/passwd  
reveillere:x:501:20:./home/reveillere:/bin/bash  
$
```



— Processus en exécution

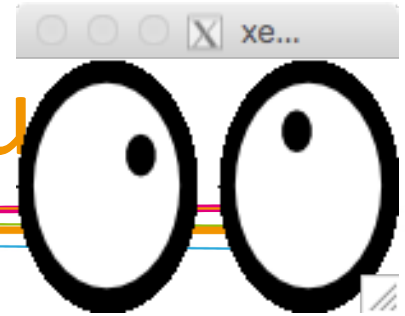
..... Processus en attente

—> Création de processus

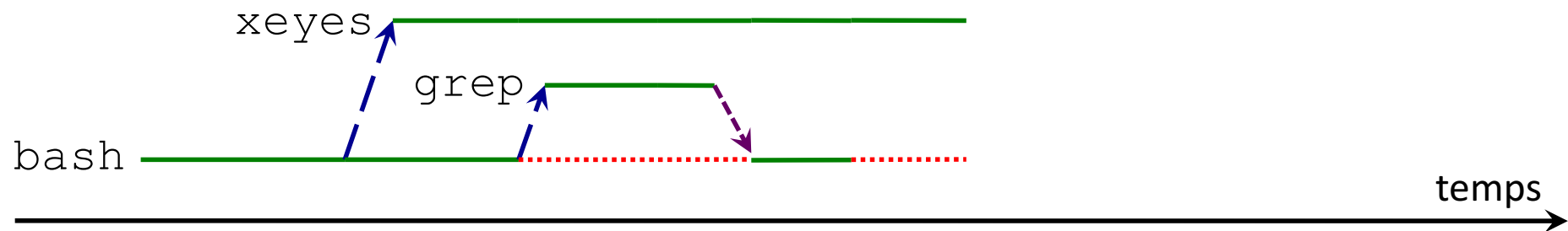
- - -> Notification de fin de processus



Attendre la fin d'un processus



```
$ xeyes &  
$ grep reveillere /etc/passwd  
reveillere:x:501:20:./home/reveillere:/bin/bash  
$ wait
```



— Processus en exécution

..... Processus en attente

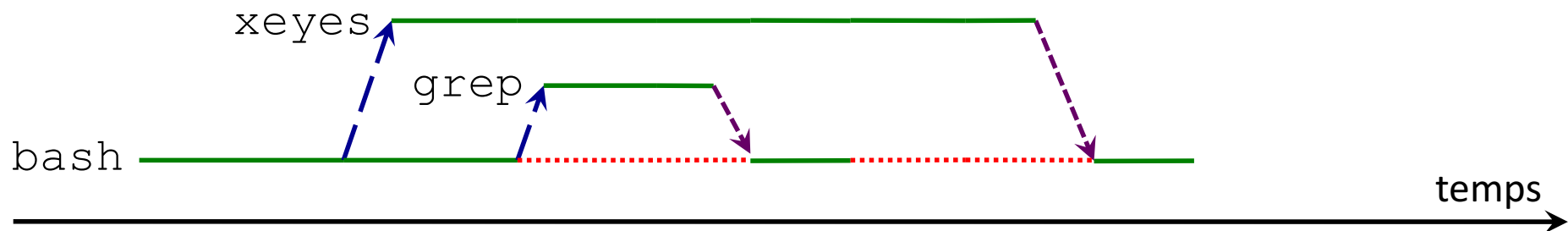
—> Création de processus

- - -> Notification de fin de processus



Attendre la fin d'un processus

```
$ xeyes &  
$ grep reveillere /etc/passwd  
reveillere:x:501:20:./home/reveillere:/bin/bash  
$ wait  
[1]+  Done                  xeyes  
$
```



— Processus en exécution

..... Processus en attente

—> Création de processus

- - -> Notification de fin de processus



Plan

1. Observer un processus
2. Processus en avant et arrière plan
3. Cycle de vie d'un processus
4. Variables et processus
5. Gestion des processus dans le système d'exploitation



Variables `bash` et processus

- ❑ Une variable est toujours locale à un processus
⇒ les modifications sont **toujours** locales

- ❑ Une variable peut être exportée chez un enfant
 - La variable et sa valeur sont recopiées chez l'enfant **à la création**
 - **Les variables du père et du fils sont ensuite indépendantes**
 - Par défaut une variable n'est pas exportée
 - Marquer une variable comme exportée : `export var`
 - Arrêter d'exporter une variable : `unset var`
(détruit aussi la variable)



Portée des variables

```
$ a="existe"  
$
```

```
#!/bin/bash  
  
b="existe"  
echo "a: $a"  
echo "b: $b"  
a="autre chose"
```

variable.sh

```
#!/bin/bash  
  
export b  
b="existe"  
echo "a: $a"  
echo "b: $b"
```

variable_exportee.sh



Portée des variables

```
$ a="existe"  
$ ./variable.sh  
a:  
b: existe  
$
```

```
#!/bin/bash  
  
b="existe"  
echo "a: $a"  
echo "b: $b"  
a="autre chose"
```

variable.sh

```
#!/bin/bash  
  
export b  
b="existe"  
echo "a: $a"  
echo "b: $b"
```

variable_exportee.sh



Portée des variables

```
$ a="existe"  
$ ./variable.sh  
a:  
b: existe  
$ export a  
$
```

```
#!/bin/bash  
  
b="existe"  
echo "a: $a"  
echo "b: $b"  
a="autre chose"
```

variable.sh

```
#!/bin/bash  
  
export b  
b="existe"  
echo "a: $a"  
echo "b: $b"
```

variable_exportee.sh



Portée des variables

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$ ./variable.sh
a: existe
b: existe
$
```

```
#!/bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
a="autre chose"
```

variable.sh

```
#!/bin/bash

export b
b="existe"
echo "a: $a"
echo "b: $b"
```

variable_exportee.sh



Portée des variables

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$ ./variable.sh
a: existe
b: existe
$ echo "a: $a - b: $b"
a: existe - b:
$
```

```
#!/bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
a="autre chose"
```

variable.sh

```
#!/bin/bash

export b
b="existe"
echo "a: $a"
echo "b: $b"
```

variable_exportee.sh



Portée des variables

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$ ./variable.sh
a: existe
b: existe
$ echo "a: $a - b: $b"
a: existe - b:
$ ./variable_exortee.sh
a: existe
b: existe
$
```

```
#!/bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
a="autre chose"
```

variable.sh

```
#!/bin/bash

export b
b="existe"
echo "a: $a"
echo "b: $b"
```

variable_exortee.sh



Portée des variables

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$ ./variable.sh
a: existe
b: existe
$ echo "a: $a - b: $b"
a: existe - b:
$ ./variable_exortee.sh
a: existe
b: existe
$ echo "b: $b"
b:
$
```

```
#!/bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
a="autre chose"
```

variable.sh

```
#!/bin/bash

export b
b="existe"
echo "a: $a"
echo "b: $b"
```

variable_exportee.sh



Variables d'environnement

- ❑ Une variable exportée s'appelle une variable d'environnement

- ❑ Certaines variables sont souvent dans l'environnement :
 - HOME : chemin absolu du répertoire de connexion
 - » `cd`, `cd ~` et `cd $HOME` sont des commandes équivalentes
 - PS1 : prompt (par défaut \$)
 - PATH : liste des répertoires de recherche des commandes
 - » Rappel : entre chaque chemin, séparateur « : »



Plan

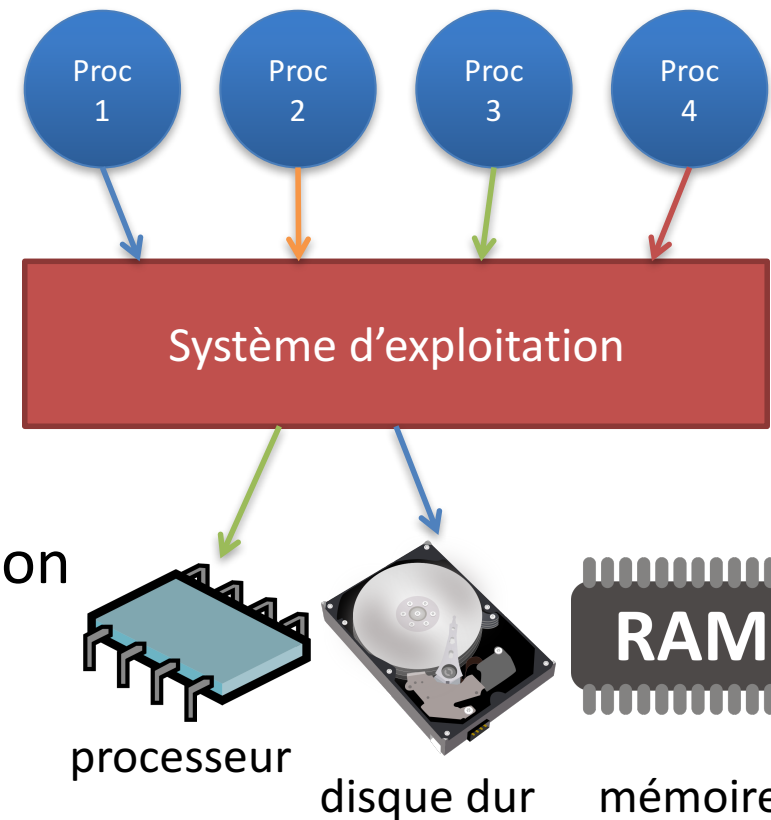
1. Observer un processus
2. Processus en avant et arrière plan
3. Cycle de vie d'un processus
4. Variables et processus
5. Gestion des processus dans le système d'exploitation



Partage de ressources

□ Ressources partagées par les processus

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



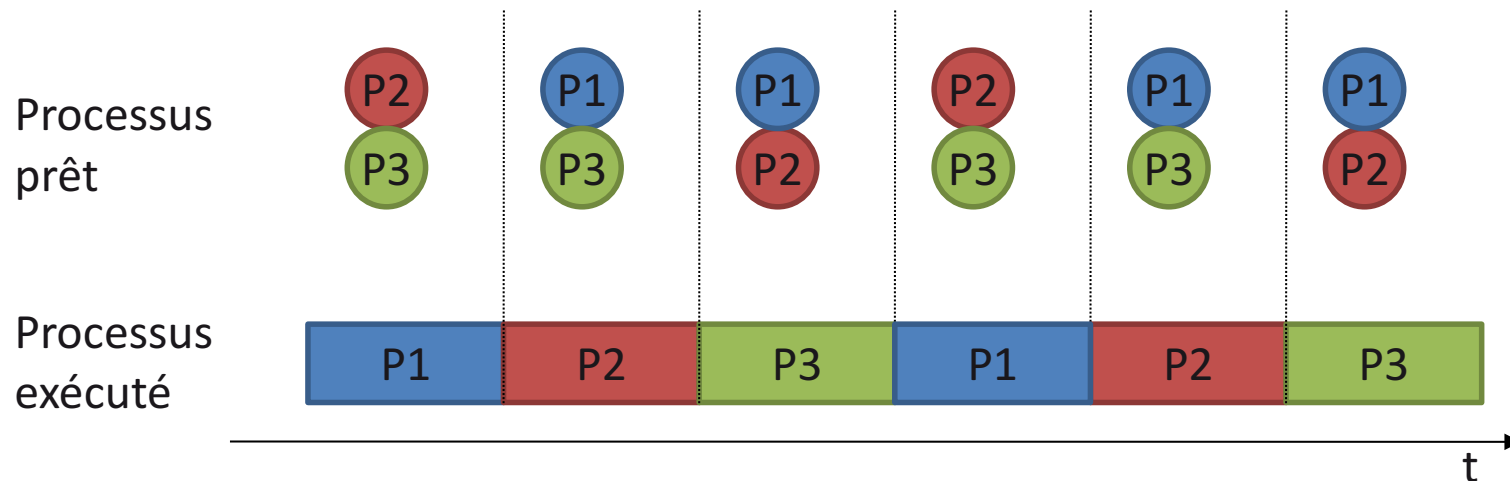
□ Gestion par le Système d'Exploitation

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



Partage du CPU

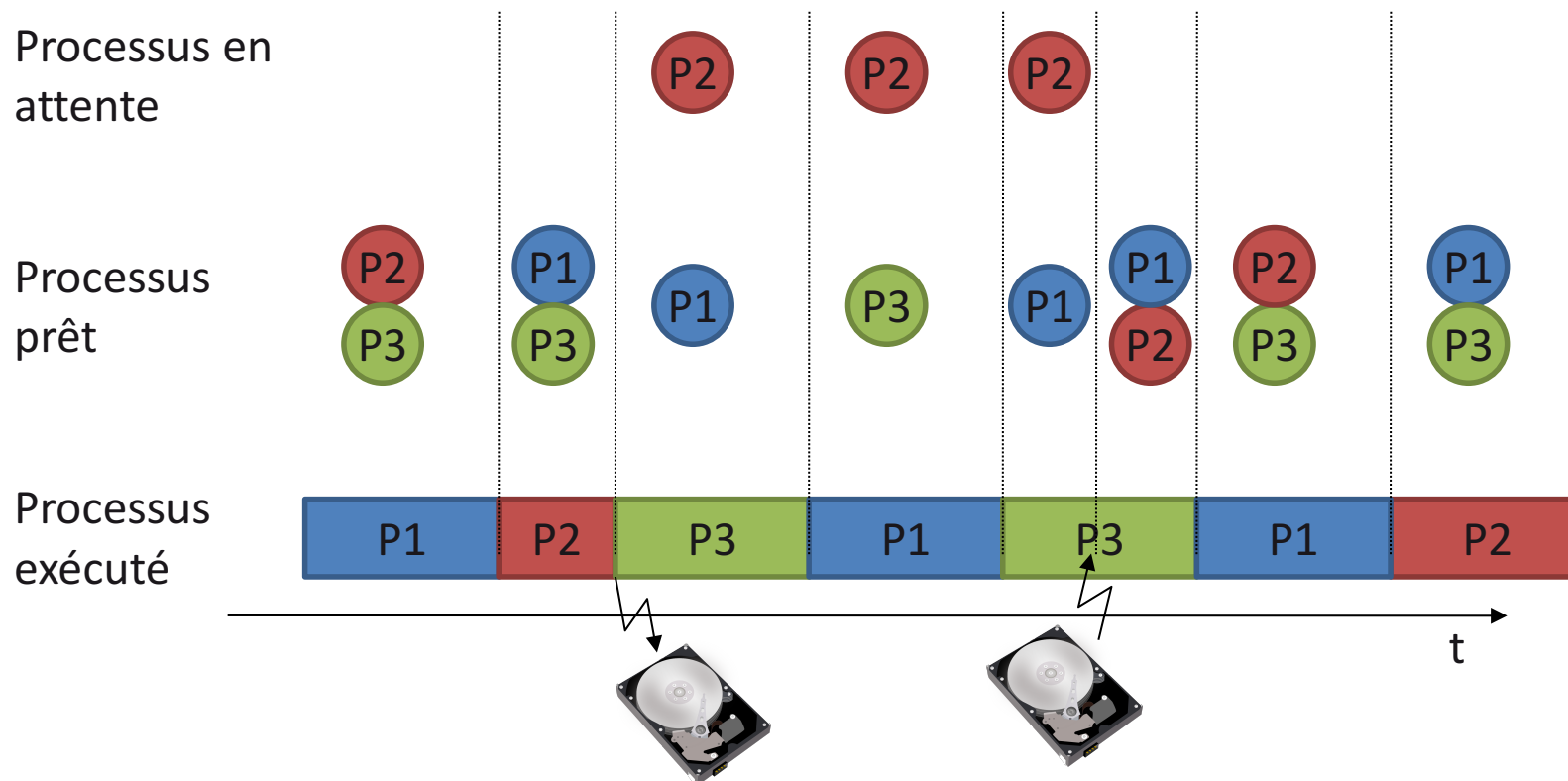
- ❑ À un instant donné, le CPU n'exécute qu'un processus
 - Les autres processus attendent
- ❑ L'**ordonnanceur** partage le CPU par « quantum de temps » (en anglais, *timeslice*)
 - À la fin du *timeslice*, l'ordonnanceur préempte le processus s'exécutant et choisit un autre processus





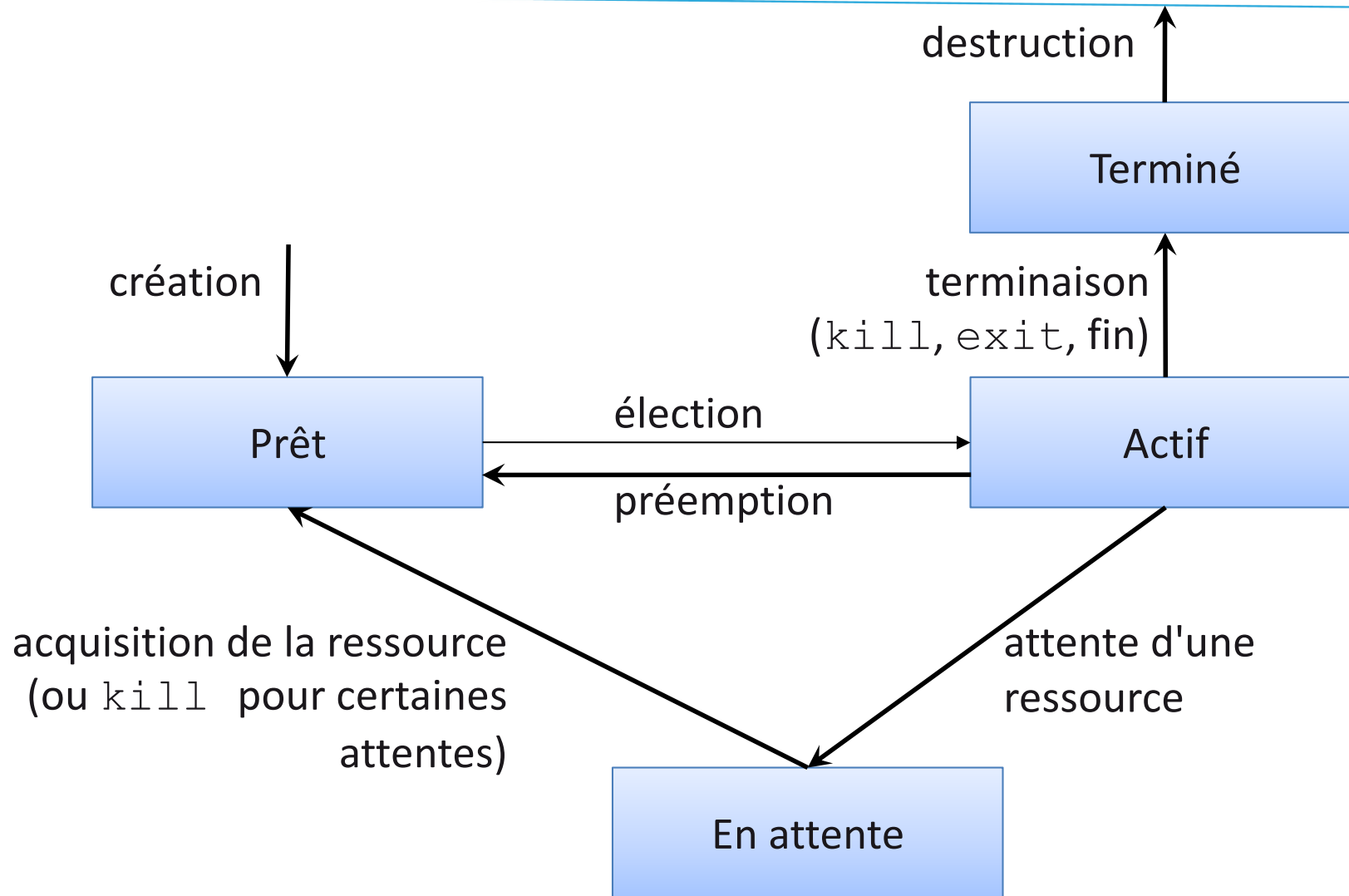
Partage du CPU et entrées/sorties

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





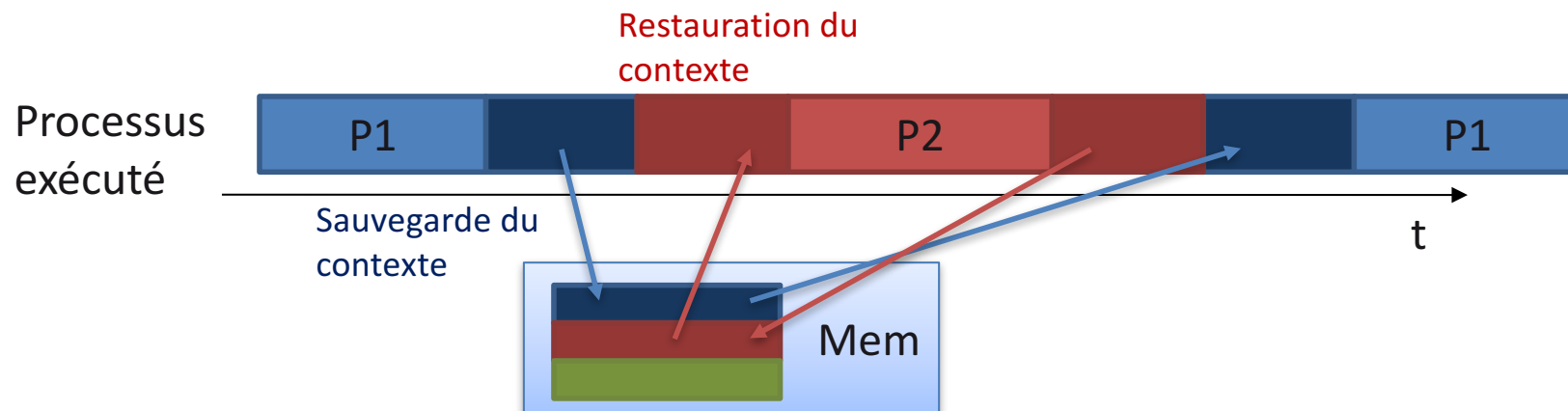
États d'un processus





Commutation de processus

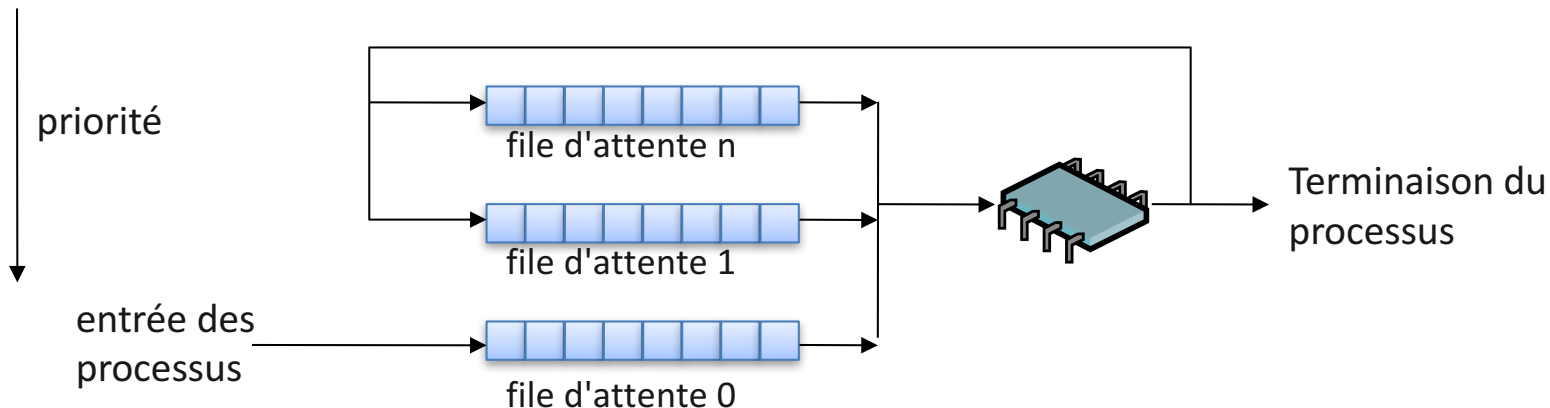
- La commutation a lieu lors de l'élection d'un processus:
 - Sauvegarde du contexte du processus évincé
 - Chargement du contexte du processus élu
- Contexte : ensemble des infos associées au processus
 - Valeur des registres
 - Informations mémoire (emplacement, etc.)





Ordonnancement de processus

- 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 un processus de la file d'attente non vide de plus haute priorité
 - Si un processus consomme tout son *timeslice* : `priorité--`
 - Régulièrement : `priorité++` pour les processus non élus





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, sortie du terminal, ...
- ❑ L'OS assure l'exclusion mutuelle
 - À tout moment, seul un processus manipule la ressource

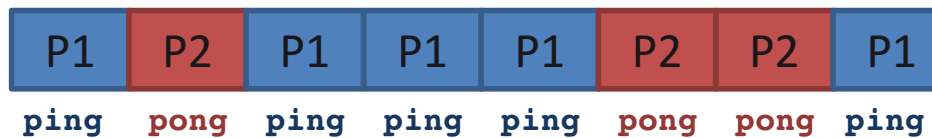
```
$ ./do_ping.sh &
./do_pong.sh
ping
pong
ping
pong
ping
pong
ping
pong
ping
pong
ping
ping
ping
ping
pong
```

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

do_ping.sh

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

do_pong.sh



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`