

# Processus

- [Les processus, à quoi ça sert ?](#)
  - [Une définition d'un processus](#)
  - [La vie intime des processus](#)
  - [Quelques caractéristiques des processus](#)
  - [Le contexte et la commutation de contexte](#)
  - [Les processus sous Unix](#)
  - [La communication entre processus](#)
  - [Les threads](#)
- 

## Les processus, à quoi ça sert ?

Ça sert à faire plusieurs activités en "*même temps*".

**Par exemple :**

Faire travailler plusieurs utilisateurs sur la même machine. Chaque utilisateur a l'impression d'avoir la machine à lui tout seul.

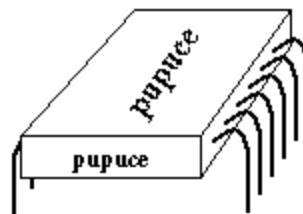
**Par exemple :**

Compiler tout en lisant son mail.

**Problème :** Un processeur ne peut exécuter **qu'une seule instruction** à la fois.

**But :** **Partager** un (ou plusieurs) processeur entre différents programmes (les processus).

Attention ! ne pas confondre **Processus** avec **Processeur** =



[Notes sur les champs d'application des processus.](#)

## Une définition d'un processus

Un processus est l'activité résultant de l'exécution d'un programme séquentiel, avec ses données, par un processeur.

## La vie intime des processus

### Allocation du processeur

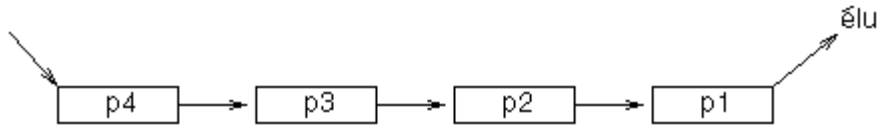
Il existe différentes stratégies :

- avec ou sans réquisition du processeur (stratégies préemptives ou non-préemptives),
- fixées à priori en utilisant des informations sur le processus et l'activité du système.

### méthode FIFO (First In, First Out)

Aussi appelé traitement par "train", par "lot" ou "batch".

Les processus accèdent au processeur, chacun à leur tour, dans l'ordre d'arrivée, et monopolisent le processeur jusqu'à leur terminaison.



⇒ Travaux courts pénalisés

⇒ Temps de Réponse fonction de la Charge du système ⇒ Stratégie indépendante du temps d'exécution des processus

Cette méthode est *non-préemptive*, c'est-à-dire qu'un processus monopolise le processeur jusqu'à sa terminaison.

Une amélioration de la stratégie FIFO est d'ordonner la file en fonction du temps estimé d'exécution des processus. Dans ce cas, le temps de réponse des travaux courts est diminué, et celui des travaux long est augmenté.



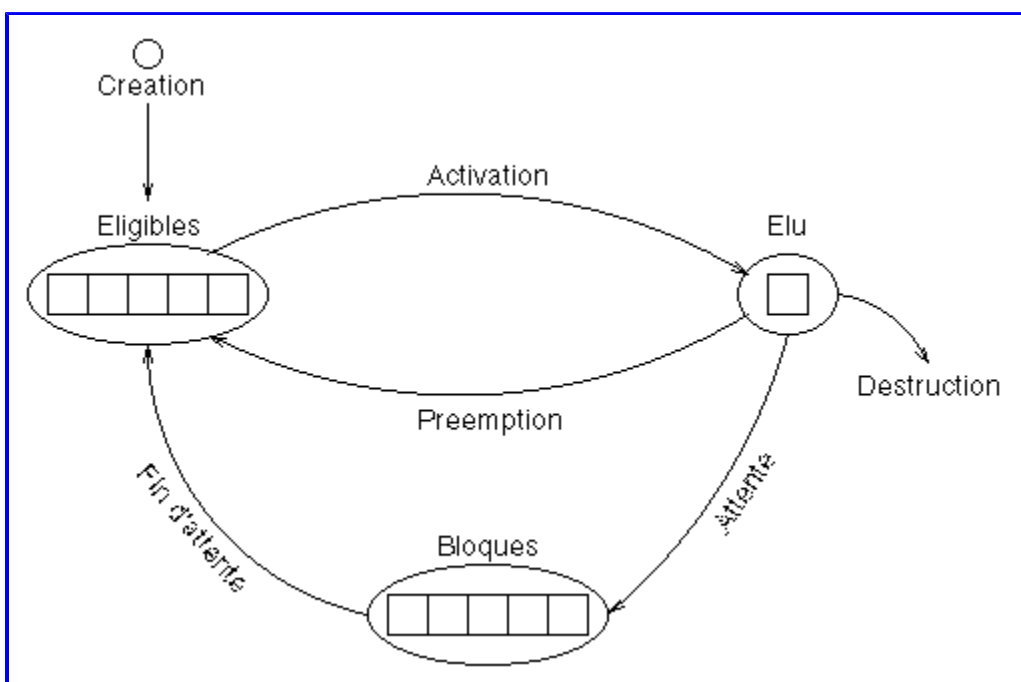
[Notes sur le traitement par lot.](#)

### Méthode du tourniquet (Round Robin)

Aussi appelé "balayage cyclique".

Les processus accèdent au processeur, chacun à leur tour, pour un temps déterminé à l'avance (le **quantum**).

Un processus en attente d'une entrée-sortie sera placée dans une file des **bloqués**.



⇒ Plus court d'abord (SJF : Shortest Job First)

⇒ Priorité = Ordonnement de la file

⇒ Privation des travaux longs

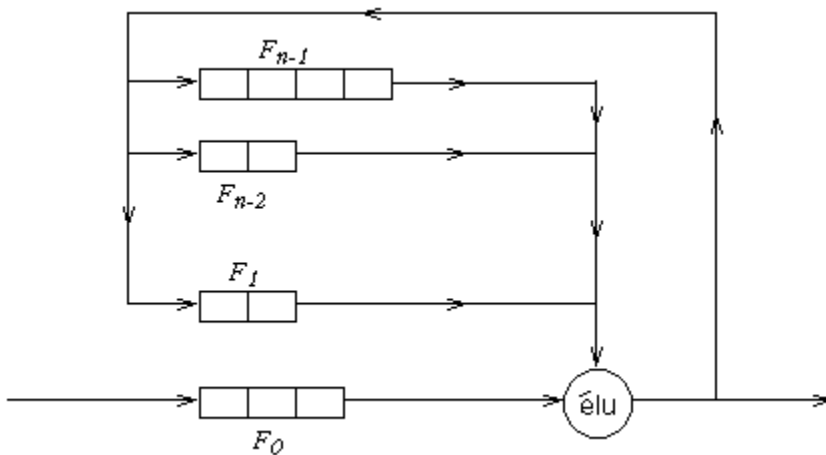
Cette méthode est *préemptive*, c'est-à-dire que le processeur est retiré à un processus au bout d'un certain quantum.

Le quantum est calculé par tâtonnements :

- si le quantum est très grand, on obtient une file d'attente simple;
- si le quantum est très petit, les processus n'auront pas le temps d'être exécuté en partie avant d'être réinsérés dans la file.

### Méthode du tourniquet multiniveaux

Avant d'accéder au processeur, les processus sont rangés dans les files correspondant à leur niveau de priorité. Un processus ne peut accéder au processeur que s'il n'existe plus de processus dans les files de plus haute priorité.



⇒ Amélioration du tourniquet simple

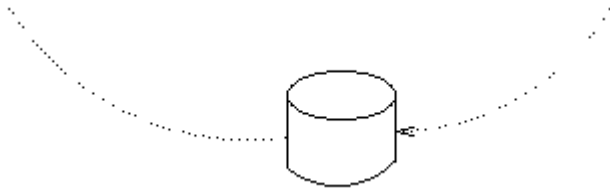
⇒ Priorité = 1 priorité différente par niveau

Les priorités peuvent être :

- *externes*, fixées avant la prise en compte du processus,
- *internes*, ajustées par le système,
- *mixtes*.

### Une amélioration du système par le swap





Un processus peut être rangé (*swapped*) sur disque s'il reste trop longtemps dans la file des bloqués.

## Quelques caractéristiques des processus

Voici un résultat possible de la commande `ps` sous Unix.

Chaque ligne concerne un processus et chaque colonne donne une caractéristique des processus. Par exemple PID est l'IDentificateur du Processus.

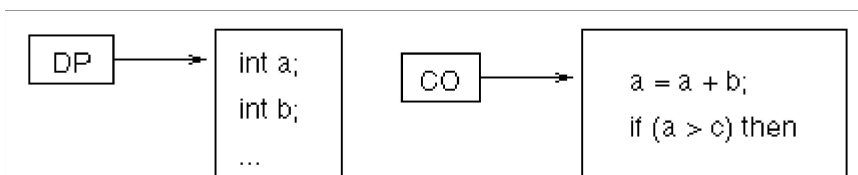
Pour plus d'information, reportez-vous aux **man pages** en tapant la commande `man ps`.

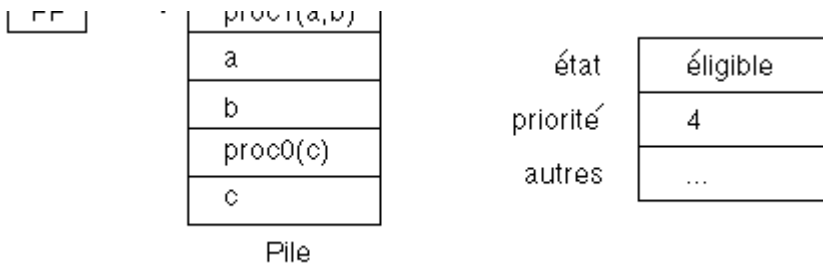
USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	START	TIME	COMMAND
billard	9343	5.4	1.3	316	756	p1	S	15:58	0:00	-local/bin/tcsh
solana	29087	2.7	7.0	3196	4212	co	S	Jan 4	81:53	lemacs
solana	10113	0.0	0.0	104	0	co	IW	Dec 22	0:01	-csh (csh)
bin	60	0.0	0.0	36	0	?	IW	Nov 15	0:01	ypbind
root	0	0.0	0.0	0	0	?	D	Nov 15	19:23	swapper
root	78	0.0	0.1	60	60	?	I	Nov 15	5:25	syslogd
ncsa	18010	0.0	0.0	1976	0	?	IW	Jan 9	0:01	/net/bin/sp

STAT	Signification
S	Sleeping <= 20s
I	Idle > 20s
W	Swapped
D	Non-interruptible

## Le contexte d'un processus

Le contexte d'un processus est l'ensemble des informations dynamiques qui représente l'état d'exécution d'un processus (e.g. où est-ce que le processus en est de son exécution).



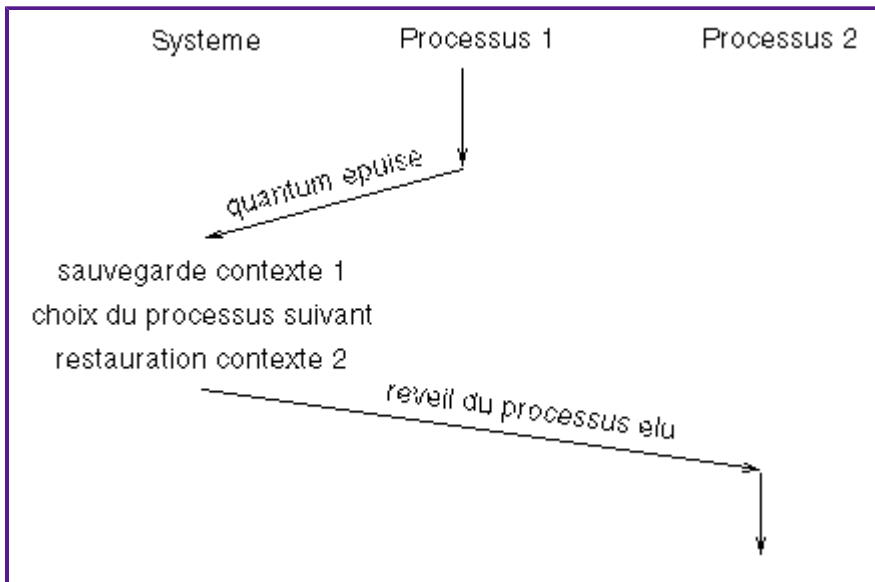


Le contexte  $\Rightarrow$  *état courant d'un processus.*

On définit aussi le *vecteur d'état* d'un processus (PSW : Program Status Word) comme l'ensemble des bits de condition, priorité, etc. au moment de la commutation de contexte.

## La commutation de contexte (context switching)

La commutation de contexte est le mécanisme qui permet au système d'exploitation de remplacer le processus élu par un autre processus éligible.



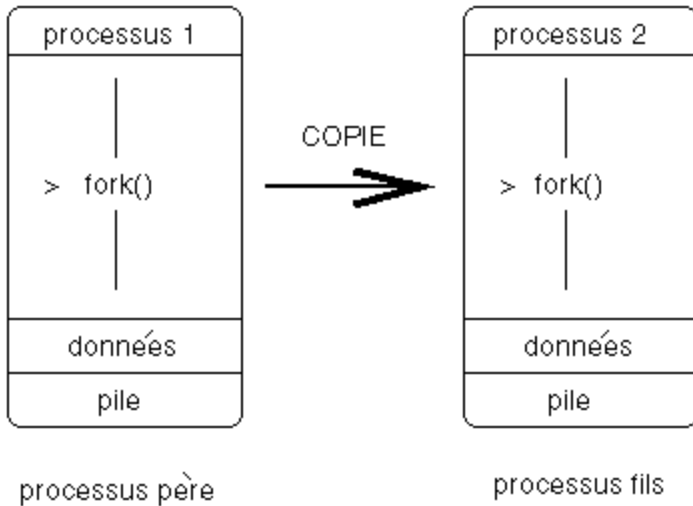
Le temps nécessaire à la commutation de contexte doit être **inférieur** au quantum.

## Les processus sous Unix

Aussi appelés processus **lourds**.

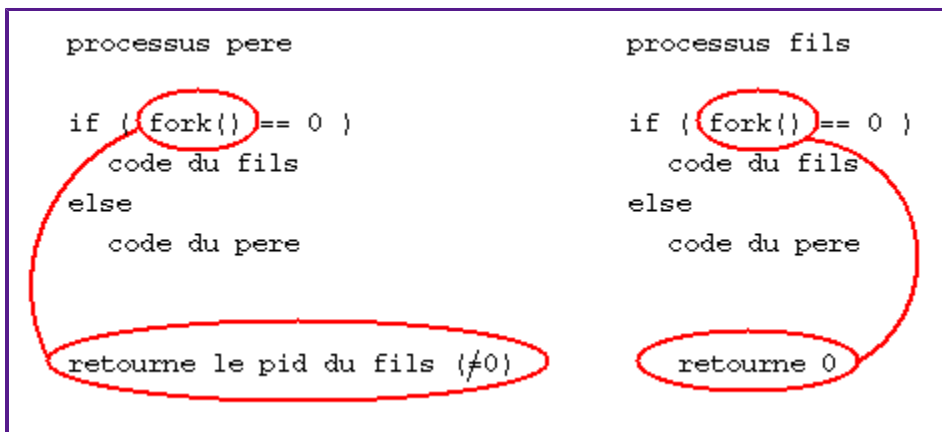
La création d'un processus :

- Sous l'interpréteur de commande (*shell*).
- Dans un programme : instruction `fork()`.



Action de `fork()` ↗

- duplication du processus père ;
- retour de la valeur **pid** (numéro du fils) dans le processus père ;
- retour de la valeur **0** dans le processus fils.



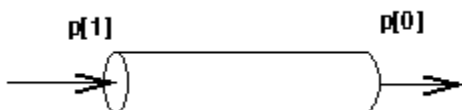
Lors du démarrage de Unix, deux processus sont créés :

- le *Swapper* (pid = 0) qui gère la mémoire;
- le *Init* (pid = 1) qui crée tous les autres processus.

## La communication entre processus

### La communication entre père et fils par *tubes* (pipe)

- Un processus hérite de son père les descripteurs de tubes.
- Un tube est composé de deux entrées (lecture / écriture).
- Chaque entrée est FIFO.
- Création d'un tube : `int pipe(p)` avec `int p[2]`.
- `p[0]` = accès en lecture ; `p[1]` = accès en écriture.



## La communication par *tubes nommés*

Un tube nommé est un fichier **spécial** créé avec la commande `mknod` ou `mkfifo`.

```
-rw-r----- 1 billard telecom 15001 fév 4 13:59 fichier_normal
prw-r----- 1 billard telecom 0 fév 4 13:58 mon_tubē
-rw-r----- 1 billard telecom 45876 fév 4 13:59 un_autre_fichier
```

La taille de `mon_tube` = 0, sauf lors de l'utilisation du tube.

Le tube s'utilise *comme un fichier normal* avec les primitives **open**, **read**, **write**, **close**.

## La communication par *IPC (Inter Process Communication)*

Les IPC font partie de l'interface Unix System V.

Les IPC comprennent :

1. les *files de messages* ;
2. la *mémoire partagée* ;
3. les *sémaphores*.

Les files de messages et la mémoire partagée sont des outils de *communication*.

Les *sémaphores* sont des outils de *synchronisation*.

### Les files de messages

1 file de message  $\equiv$  1 boîte aux lettres.

Un processus peut déposer et retirer des messages. Les messages sont typés.

A	B	C	A	C
msg 1	msg 2	msg 3	msg 4	msg 5

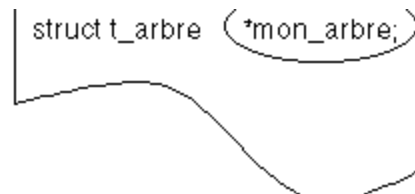
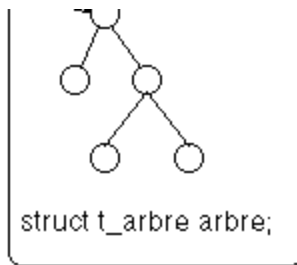
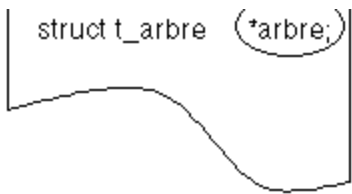
Un processus peut retirer :

- le premier message de la file ( $\forall$  type) ;
- le premier message de type  $t$  ;
- le premier message de type  $\leq t$  ( $t$  étant une valeur entière  $> 0$ ).

### La mémoire partagée

1 mémoire partagée  $\equiv$  1 espace d'adressage commun à plusieurs processus.

Un processus peut lire et écrire en mémoire partagée, comme s'il s'agissait de ses propres variables.



## Les IPC - Inconvénients

Table des IPC (donnée par la commande `ipcs`).

```

Message Queues:
T      ID      KEY          MODE          OWNER          GROUP
q  28050 19466732  --rw-rw-rw-  cbronner      sys2
Shared Memory:
T      ID      KEY          MODE          OWNER          GROUP
m  10601      4353  --rw-rw-rw-  marchal      sys2
Semaphores:
T      ID      KEY          MODE          OWNER          GROUP
s     811      8765  --ra-ra-ra-  marchal      sys2
  
```

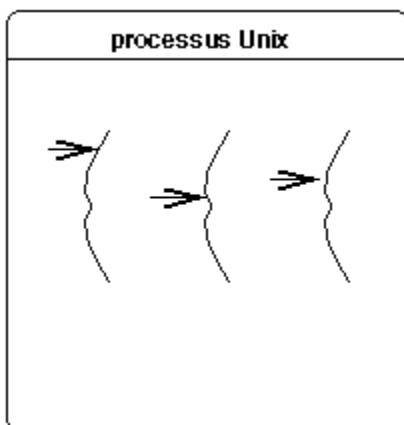
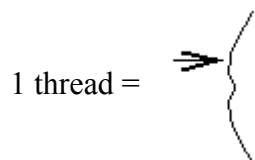
Les IPC font partie du système d'exploitation.

Chaque opération sur un IPC implique donc un appel système, très **couteux**, *i.e.* **lent**.

## Les threads - processus légers

**Idée** : plusieurs threads à l'intérieur du même processus.

Chaque thread accède au même segment de données (donc aux mêmes variables).



Une thread qui interagit avec une autre au sein du même processus **n'utilise pas** le système d'exploitation.

⇒ une thread est plus *légère* à gérer et sa gestion peut être *personalisée*.

La commutation de contexte est plus simple entre threads.

États uniques pour chaque thread :

- identificateur de thread ;
- état des registres ;
- pile ;
- masques de signaux (décrivent à quels signaux la thread répond) ;
- priorité ;
- données privées à la thread.

**Attention** : par défaut, la méthode d'allocation du processeur est NON-préemptive dans les anciens systèmes, et préemptive dans tous les systèmes récents.

---



[Retour au sommaire.](#)