

Les processus



Julien Baste

IUT de Lille

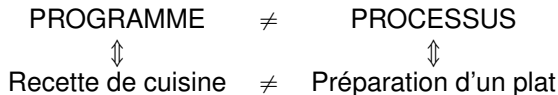
Séance 03

2025/2026

Programme → suite d'instructions que le système doit faire accomplir au processeur pour résoudre un problème particulier.

- Ces instructions sont stockées dans un fichier.

Processus → déroulement (*l'exécution*) d'un programme par le système dans un environnement particulier.



Un processus peut démarrer d'autres processus.

- Comme pour le système de fichiers, on a une notion d'**arborescence des processus**.

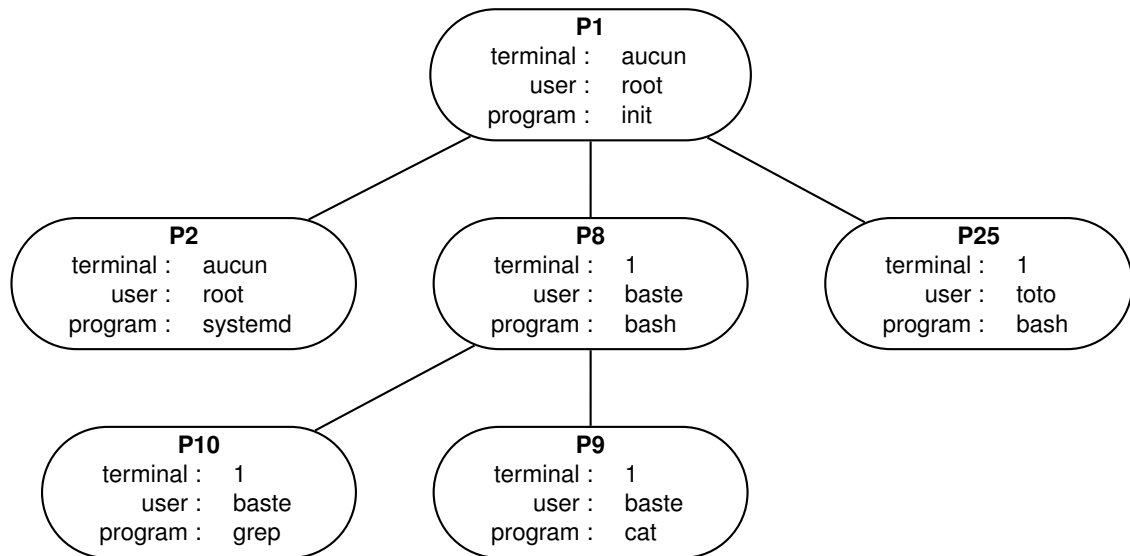
Le **processus initial** s'appelle `init`.

- C'est lui qui est lancé par le bios ou l'uefi au lancement du système d'exploitation

On distingue 2 types de processus :

- interactifs : associé à un terminal particulier
- non-interactifs (*daemons*) : attaché à aucun terminal

Le shell démarre **un processus interactif pour chaque commande** donnée par l'utilisateur.



- Le noyau maintient une table pour gérer l'ensemble des processus
- Chaque processus est identifié par un index dans cette table

son numéro d'identification ou **PID**

- Chaque entrée de la table correspond aux informations sur ce processus :
 - le numéro d'identification du processus père **PPID**
 - l'identifiant de l'utilisateur qui exécute le processus **UID**
 - l'identifiant du groupe de l'utilisateur qui exécute le processus **GID**
 - le répertoire courant **cwd**
 - la liste des fichiers utilisés par le processus
 - le masque de création des fichiers **umask**
 - la taille maximale des fichiers que ce processus peut créer **ulimit**
 - le terminal de contrôle associé
 - la zone mémoire associée code, données, pile et tas
- Plus d'informations : `proc(5)`, `ps(1)`

Un processus peut être lancé :

- En **avant plan** (foreground)
 - Mode dit synchrone : Attend que le fils ait fini pour continuer
- En **tâche de fond** (background)
 - Mode dit asynchrone : continue sans se préoccuper de son fils

En shell, le mode par défaut est le mode synchrone. Il est possible de changer ce mode en terminant la commande par :

- des **points-virgules** (;)
 - Lance la commande en avant plan
- des **esperluètes** (&)
 - Lance la commande en tâche de fond
- des **tubes** (|)
 - Lance les commandes en parallèle en les chaînant

Tous les processus gère une table stockant le nom des différents fichiers qu'ils utilisent. Chaque index de cette table est appelé un **descripteur de fichiers**.

Par convention les trois premiers descripteurs correspondent à :

0 - l'**entrée standard** :

- Ouvert en mode lecture
- Permet à l'utilisateur de donner des informations au processus

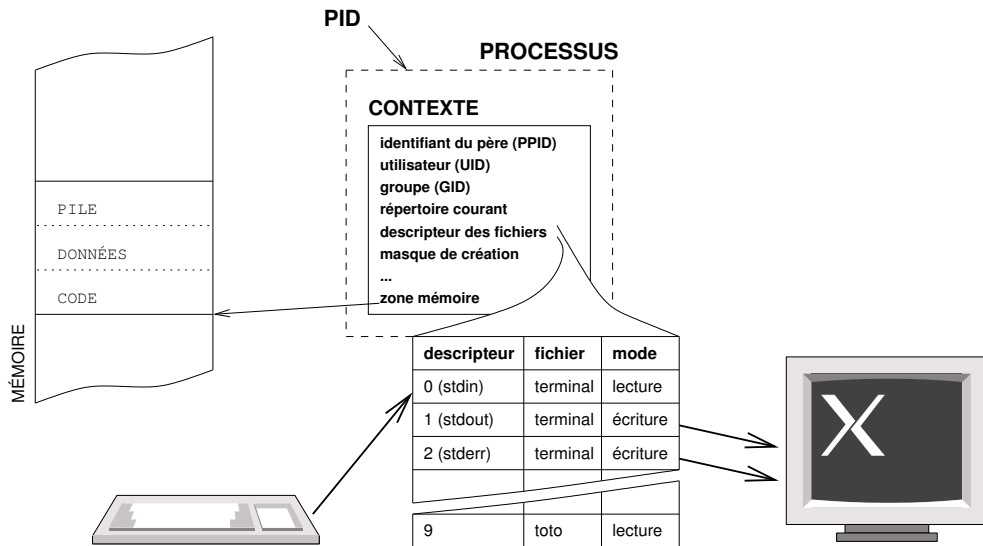
1 - la **sortie standard** :

- Ouvert en mode écriture
- Permet au processus de donner des informations à l'utilisateur

2 - la **sortie d'erreur** :

- Ouvert en mode écriture
- Permet au programme d'envoyer un message d'erreur à l'utilisateur

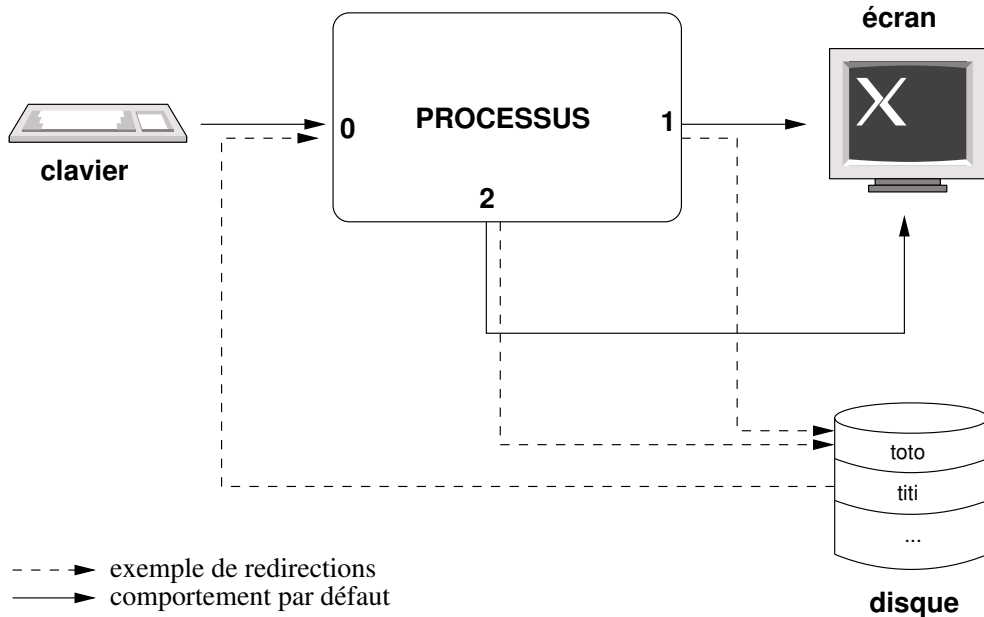
Par défaut ces trois descripteurs désignent le **terminal courant**.



En shell, il est possible de modifier les fichiers identifiés par les descripteurs :

- Redirection de la sortie standard avec le caractère plus grand «>» :
 - `commande > fichier`
Si le fichier n'existe pas, il est créé par le shell et s'il existe déjà le shell détruit son **contenu** pour le **remplacer** par la sortie de la commande
 - `commande >> fichier`
Si le fichier n'existe pas, il est créé par le shell et s'il existe déjà la sortie de la commande est **ajoutée à la fin du fichier**.
- Redirection de l'entrée standard avec le caractère plus petit «<» :
 - `commande < fichier`
La commande lit ses données dans le fichier.

Redirections : exemple



$\langle n \rangle > \langle \text{fichier} \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en écriture, vers $\langle \text{fichier} \rangle$.
$\langle n \rangle > > \langle \text{fichier} \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en écriture, vers $\langle \text{fichier} \rangle$, mais écrira à la suite du fichier sans détruire les données préalablement contenues dans ce fichier.
$\langle n \rangle > \& \langle m \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en écriture, sur le même fichier que le descripteur numéro $\langle m \rangle$.
$\langle n \rangle < \langle \text{fichier} \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en lecture, vers $\langle \text{fichier} \rangle$.
$\langle n \rangle < < \langle \text{marque} \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en lecture, vers les lignes suivantes jusqu'à ce que la $\langle \text{marque} \rangle$ soit lue.
$\langle n \rangle < \& \langle m \rangle$	Le descripteur numéro $\langle n \rangle$ pointe, en lecture, sur le même fichier que le descripteur numéro $\langle m \rangle$. Ainsi, $\langle n \rangle$ et $\langle m \rangle$ seront dirigés vers le même fichier.

Note : Il est possible de mettre autant de redirections que voulu sur une ligne de commandes.

Pour des tâches complexes, on a besoin de plusieurs commandes successives.

Idée :

- Executer la commande 1 et écrire son résultat dans un fichier
- Executer la commande 2 qui lit le fichier le traite et produit un nouveau résultat stocké dans un deuxième fichier
- Executer la commande 3 qui lit ce nouveau fichier et retourne son résultat

Problème :

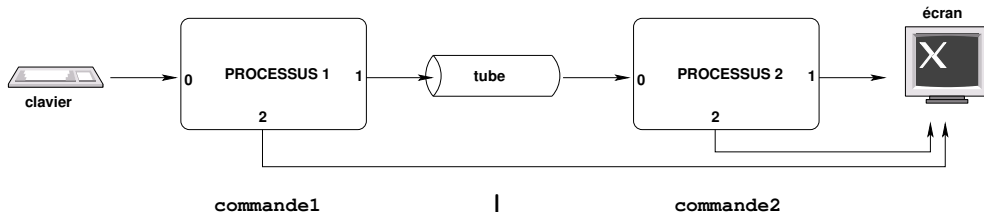
- Pour s'exécuter, la commande 2 doit attendre que la commande 1 se termine
- Que deviennent les fichiers intermédiaires ?

Il est possible d'avoir plusieurs processus fonctionnant en **parallèle** qui communiquent entre eux par le biais de **tubes** (pipes).

Rappel : Un tube est un fichier.

Le principe est assez simple :

- Le processus redirige sa sortie standard vers l'entrée d'un **tube**
- La sortie du tube est redirigée vers l'entrée standard d'un autre processus.



L'utilisation des tubes sera réalisée par une commande de la forme :

```
commande1 | commande2 | ... | commandeN
```

Ce mécanisme est une des forces d'UNIX :

- un ensemble de petits programmes **fiables** qui communiquent entre eux via le système d'exploitation.

Il existe de nombreuses commandes UNIX qui profitent de ce genre de communication, notamment les **filtres**.

Filtres : Programmes qui lisent des données sur l'entrée standard, les modifient et envoient le résultat sur la sortie standard

<code>cat</code>	retourne les lignes lues sans modification.
<code>cut</code>	ne retourne que certaines parties de chaque ligne lue.
<code>grep</code>	retourne uniquement les lignes lues qui correspondent à un modèle particulier ou qui contiennent un mot précis.
<code>head</code>	retourne les premières lignes lues.
<code>more</code>	retourne les lignes lues par bloc (dont la taille dépend du nombre de lignes affichables par le terminal) en demandant une confirmation à l'utilisateur entre chaque bloc.
<code>sort</code>	trie les lignes lues.
<code>tail</code>	retourne les dernières lignes lues.
<code>tee</code>	envoie les données lues sur la sortie standard ET dans un fichier passé en paramètre.
<code>tr</code>	remplace des caractères lus par d'autres.
<code>uniq</code>	supprime les lignes identiques.
<code>wc</code>	retourne le nombre de caractères, mots et lignes lus.
<code>sed</code>	édite le texte lu (requêtes <code>ed</code> comme avec la directive « : » de <code>vi</code>).

→ Chacune de ces commandes possède de nombreuses options décrites dans le manuel.

