Introduction aux systèmes informatiques

Codage et Systèmes d'exploitation

Pôle ASR – Module M1101 – Semestre 1

Bruno BEAUFILS

⟨bruno.beaufils@univ-lille.fr⟩

https://beaufils.u-lille.fr

Université de Lille, IUT « A », Département informatique

Année 2020/2021

Systèmes d'exploitations

Définitions

Un **programme** est une suite d'instructions que le système doit faire accomplir au processeur pour résoudre un problème particulier.
Ces instructions sont stockées dans un fichier.

Définition

Un **processus** correspond au déroulement (*l'exécution*) d'un programme par le système dans un environnement particulier.

$$\begin{array}{ccc} \text{PROGRAMME} & \neq & \text{PROCESSUS} \\ \Updownarrow & & \Updownarrow \\ \text{Recette de cuisine} & \neq & \text{Préparation d'un plat} \end{array}$$

Définitions

Un **programme** est une suite d'instructions que le système doit faire accomplir au processeur pour résoudre un problème particulier.
Ces instructions sont stockées dans un fichier.

Définition

Un **processus** correspond au déroulement (*l'exécution*) d'un programme par le système dans un environnement particulier.

PROGRAMME $\neq$ PROCESSUS

$\Updownarrow$ $\Updownarrow$

Recette de cuisine $\neq$ Préparation d'un plat

Définitions

Un **programme** est une suite d'instructions que le système doit faire accomplir au processeur pour résoudre un problème particulier.
Ces instructions sont stockées dans un fichier.

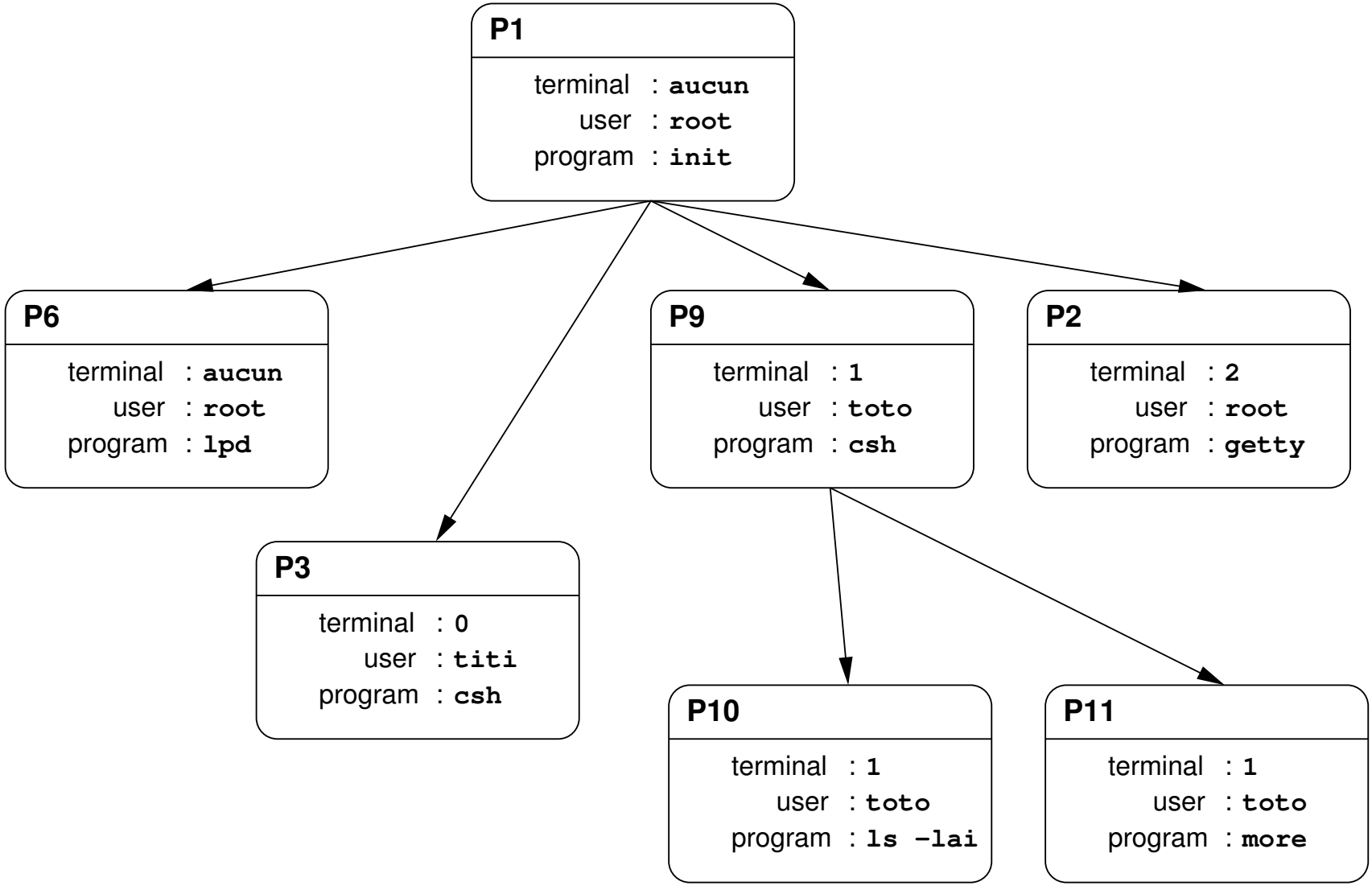
Définition

Un **processus** correspond au déroulement (*l'exécution*) d'un programme par le système dans un environnement particulier.

$$\begin{array}{ccc} \text{PROGRAMME} & \neq & \text{PROCESSUS} \\ \Updownarrow & & \Updownarrow \\ \text{Recette de cuisine} & \neq & \text{Préparation d'un plat} \end{array}$$

Généralités

- Chaque processus peut lui même démarrer d'autres processus ; dans ce cas le créateur est appelé le père et les processus qu'il a créé sont appelés ses enfants.
  Notion d'arborescence des processus
- Au démarrage du système il n'existe qu'un seul processus qui est donc l'ancêtre de tous les autres (il exécute le programme `init`).
  Son rôle est de créer 2 type de processus :
  - **interactifs** associés à un terminal particulier
  - **non-interactifs** (*daemons*) rattachés à aucun terminal
- Les processus des utilisateurs sont démarrés par un processus interactif qui exécute un programme particulier : un interpréteur de commandes (*shell*).
- **Le shell démarre un processus pour chacun des ordres (commandes) de l'utilisateur associé.**


P1
terminal : aucun
user : root
program : init
P6
terminal : aucun
user : root
program : lpd
P3
terminal : 0
user : titi
program : csh
P9
terminal : 1
user : toto
program : csh
P2
terminal : 2
user : root
program : getty
P10
terminal : 1
user : toto
program : ls -lai
P11
terminal : 1
user : toto
program : more

Contexte d'exécution

- 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)`

Modes de fonctionnement

Un processus peut

- attendre la fin de son fils pour continuer ..... **avant plan** (*foreground*)
  mode synchrone : les processus s'exécutent en *séquence*
- ne pas attendre la fin de son fils pour continuer .. **tâche de fond** (*background*)
  mode asynchrone : les processus s'exécutent en *parallèle*

Pour chaque commande exécutée le shell crée un nouveau processus.

- Par défaut en mode synchrone
- Les commandes peuvent être séparées par :
  - des **points-virgules** ;
    Attendre la fin d'une commande avant de passer à la suivante
  - des **esperluètes** &
    Ne pas attendre la fin d'une commande pour passer à la suivante
  - des **tubes** |
    Démarrer les commandes en parallèle en les chaînant

- Pour lancer une commande en avant-plan il suffit de taper cette commande :

```
$ commande
... résultat de la commande
$
```

➥ Ce mode est le mode par défaut dans les shells.

- Pour lancer une commande en tâche de fond, il faut faire suivre cette commande par le caractère esperluète « `&` » :

```
$ commande &
[1] 31343
$
... résultat de la commande
```

Le Bourne shell (`sh`) affiche un numéro de tâche (*job*) entre crochets puis le PID du processus créé avant de rendre la main à l'utilisateur.

Entrées/Sorties

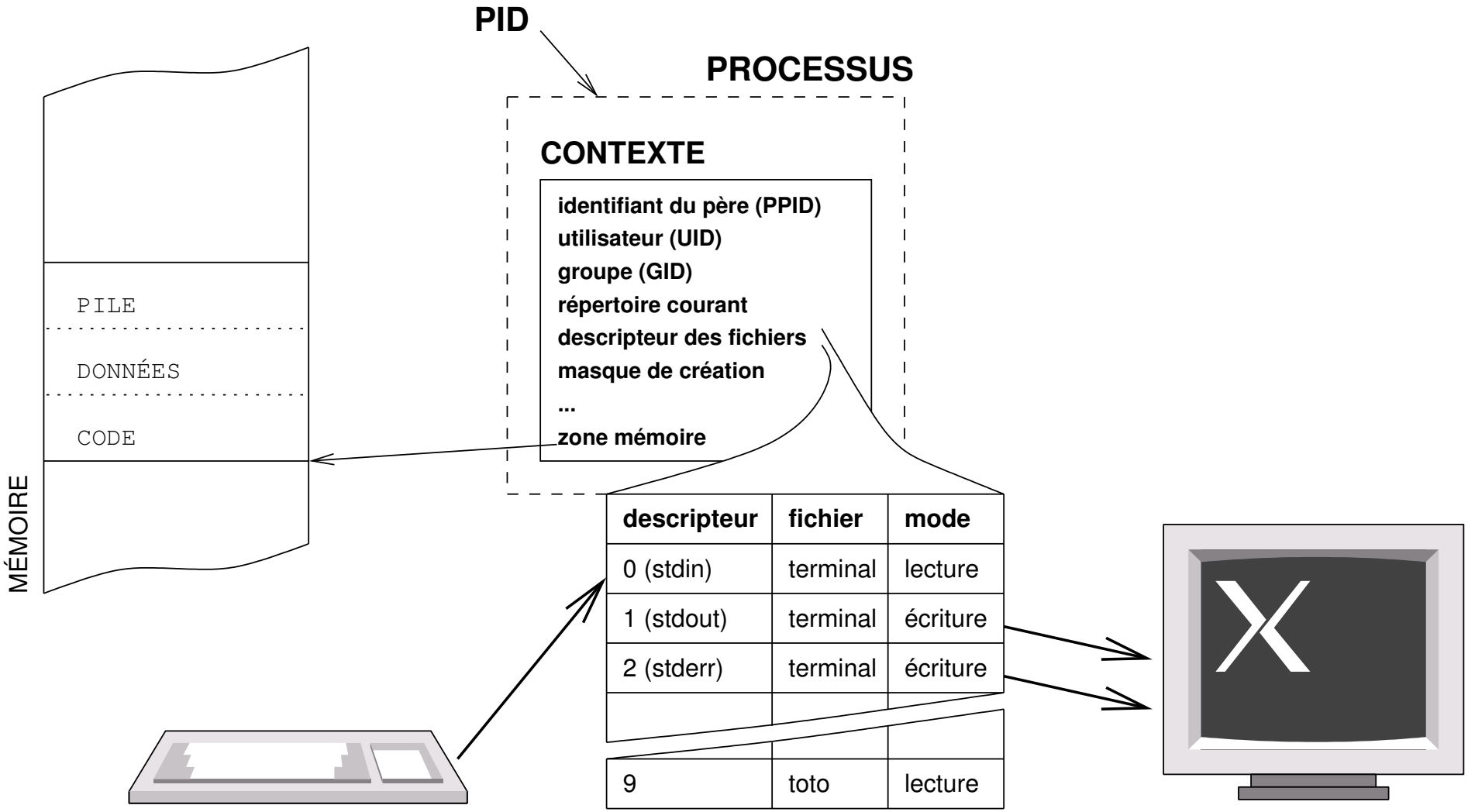
Tous les processus gère une table stockant le nom des différents fichiers qu'ils utilisent. Chaque index de cette table est appelé un *descripteur de fichiers*.

Par convention les trois premiers descripteurs correspondent à :

1. **l'entrée standard** :
   si le programme exécuté par le processus a besoin de demander des informations à l'utilisateur il les lira dans ce fichier (par défaut c'est le terminal en mode lecture).

2. **la sortie standard** :
   si le programme a besoin de donner des informations à l'utilisateur il les écrira dans ce fichier (par défaut c'est le terminal en mode écriture).

3. **la sortie d'erreur** :
   si le programme a besoin d'envoyer un message d'erreur à l'utilisateur il l'écrira dans ce fichier (par défaut c'est le terminal en mode écriture).

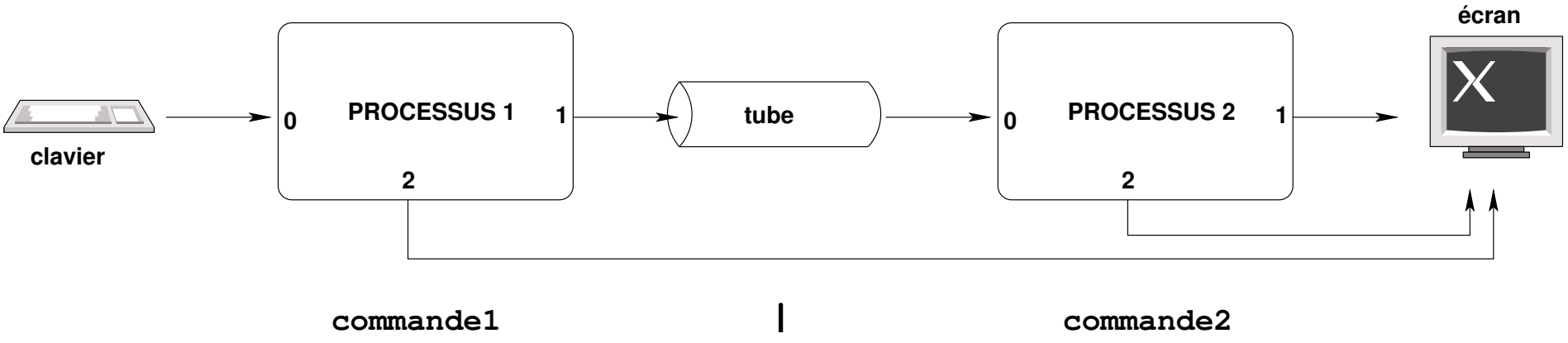

PID
PROCESSUS
CONTEXTE
identifiant du père (PPID)
utilisateur (UID)
groupe (GID)
répertoire courant
descripteur des fichiers
masque de création
...
zone mémoire
PILE
DONNÉES
CODE
MÉMOIRE
descripteur
fichier
mode
0 (stdin)
terminal
lecture
1 (stdout)
terminal
écriture
2 (stderr)
terminal
écriture
9
toto
lecture

Communication inter-processus

Il est possible d'avoir plusieurs processus fonctionnant en *parallèle* qui communiquent entre eux par le biais de **tubes** (*pipes*). Le système assure alors la synchronisation de l'ensemble des processus ainsi lancés.

Le principe est assez simple :

> *La sortie standard d'un processus est redirigée vers l'entrée d'un tube dont la sortie est dirigée vers l'entrée standard d'un autre processus.*



Le lancement concurrent de processus communiquant deux par deux par l'intermédiaires des tubes sera réalisé 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* :

Définition

Les filtres sont des programmes qui lisent des données sur l'entrée standard, les modifient et envoient le résultat sur la sortie standard

Quelques filtres

`cat`	retourne les lignes lues sans modification.
`cut`	ne retourne que certaines parties de chaque lignes lues.
`grep`	retourne uniquement les lignes lues qui correspondent à un modèle particulier ou qui contiennent un mot précis.
`head`	retourne les premières lignes lues.
`more`	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.
`sort`	trie les lignes lues.
`tail`	retourne les dernières lignes lues.
`tee`	envoie les données lues sur la sortie standard **ET** dans un fichier passé en paramètre.
`tr`	remplace des caractères lus par d'autres.
`uniq`	supprime les lignes identiques.
`wc`	retourne le nombre de caractères, mots et lignes lus.
`sed`	édite le texte lu (requêtes `ed` comme avec la directive « `:` » de `vi`).

➜ Chacune de ces commandes possèdent de nombreuses options décrites dans le manuel.

```
bash-2.08$ getent passwd | grep beaufils | tee /tmp/out | cut -d:
Bruno.BEAUFILS
```

```
bash-2.08$ cat /tmp/out
beaufils:x:1000:1000:Bruno.BEAUFILS:/home/ens/beaufils:/bin/bash
```

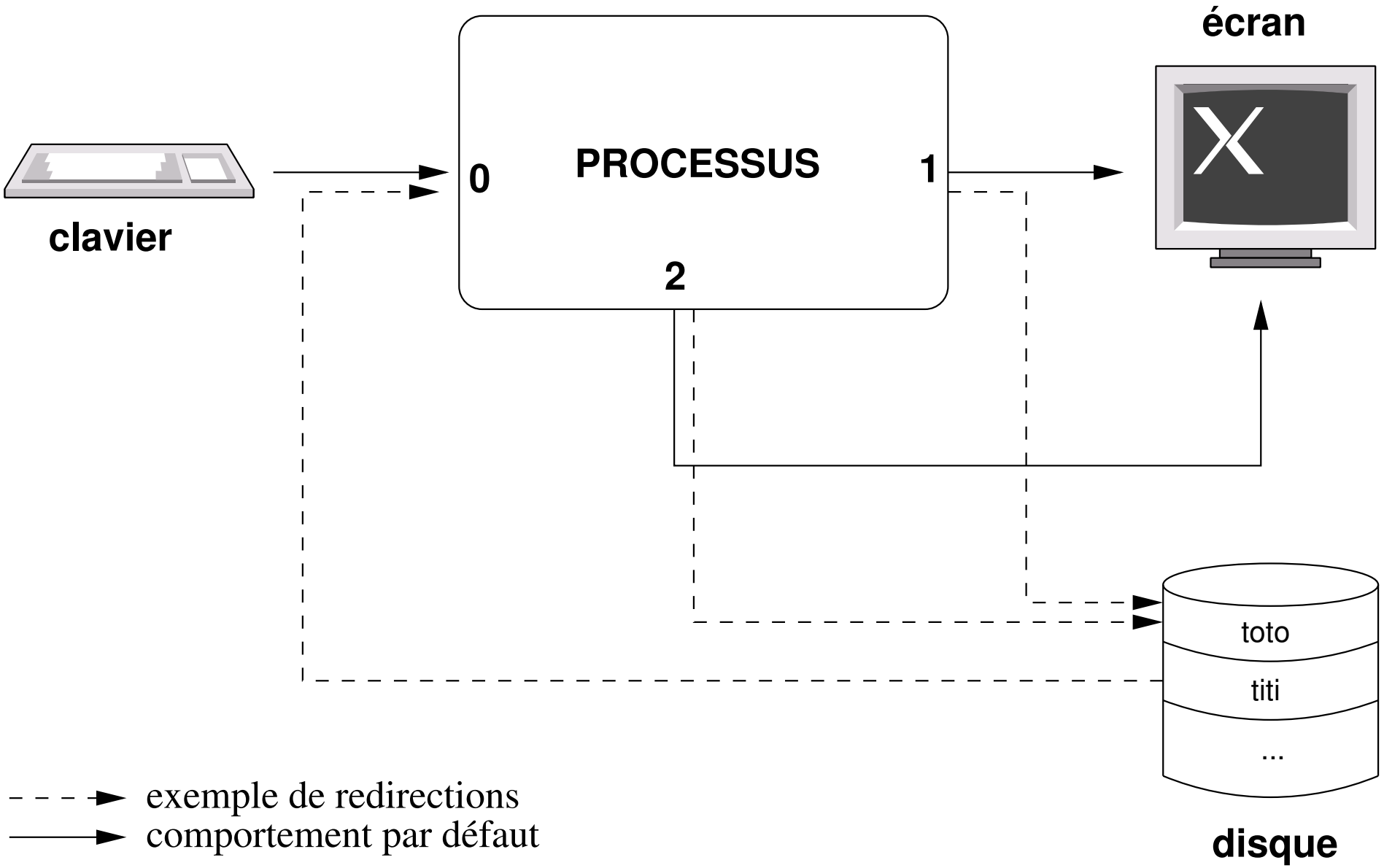
```
bash-2.08$ ls -l /media/
total 16
drwxr-xr-x  2 root root 4096 2002-12-30 19:28 cdrom
drwxr-xr-x  2 root root 4096 2002-12-30 19:28 floppy
drwxr-xr-x  2 root root 4096 2003-05-16 08:08 usb
drwxr-xr-x  2 root root 4096 2003-11-12 01:43 zip

bash-2.08$ ls -l /media | grep usb | cut -d\  -f1 | cut -c 5-7
r-x
```

```
bash-2.08$ getent passwd \
| grep Laurent | tee /tmp/t1 \
| cut -d: -f 5 | tee /tmp/t2 \
| sort -t. -k2 -r | tee /tmp/t3 \
| tr . ' '

bash-2.08$ cat /tmp/t1
blondel:x:1447:1020:Laurent.BLONDEL:/home/ens_ext/blondel:/bin/bash
behaguel:x:1141:1015:Laurent.BEHAGUE:/home/iutfi2/behaguel:/bin/bash
mulierl:x:1331:1014:Laurent.MULIER:/home/iupqepi3/mulierl:/bin/bash

bash-2.08$ cat /tmp/t2
Laurent.BLONDEL
Laurent.BEHAGUE
Laurent.MULIER

bash-2.08$ cat < /tmp/t3
Laurent.MULIER
Laurent.BLONDEL
Laurent.BEHAGUE
```

Redirections

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 crée 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 crée 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 petit « **<** » :
  - `commande < fichier`
    La commande lit ses données dans le fichier.


écran
PROCESSUS
0
1
2
clavier
toto
titi
...
disque
exemple de redirections
comportement par défaut

Syntaxe générale des redirections

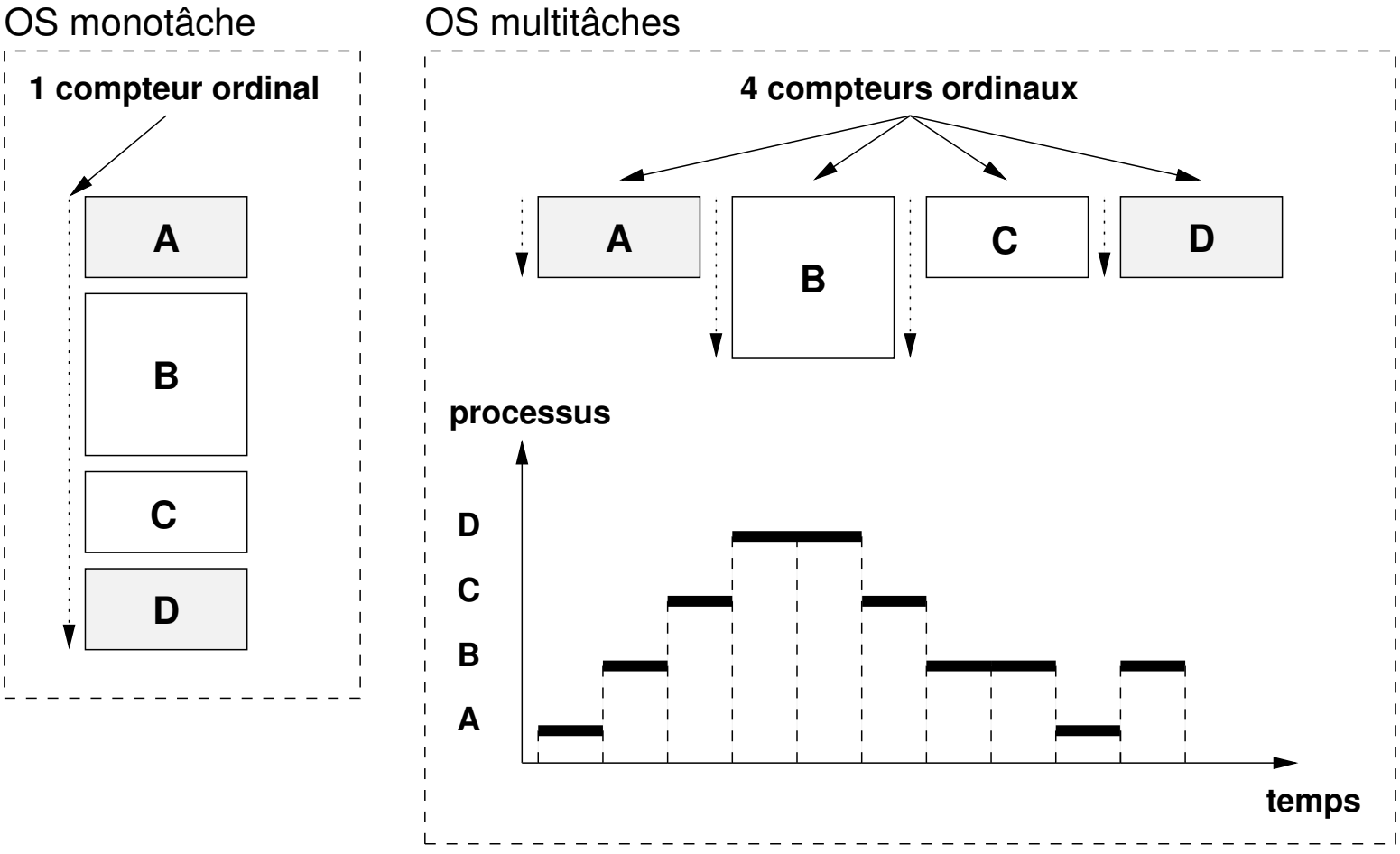
⟨*n*⟩`<`⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ en lecture vers ⟨*fichier*⟩.
⟨*n*⟩`>`⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ en écriture vers ⟨*fichier*⟩.
⟨*n*⟩`<<`⟨*marque*⟩	redirige le descripteur numéro ⟨*n*⟩ en lecture vers les lignes suivantes jusqu'à ce que la ⟨*marque*⟩ soit lue.
⟨*n*⟩`>>`⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ à la fin de ⟨*fichier*⟩ sans détruire les données préalablement contenues dans ce fichier.
⟨*n*⟩`<&`⟨*m*⟩	duplique le descripteur numéro ⟨*n*⟩ sur le descripteur numéro ⟨*m*⟩ en lecture, ainsi ⟨*n*⟩ et ⟨*m*⟩ seront dirigés vers le même fichier.
⟨*n*⟩`>&`⟨*m*⟩	duplique le descripteur numéro ⟨*n*⟩ sur le descripteur numéro ⟨*m*⟩ en écriture.

➥ Il est possible de mettre autant de redirections que voulues sur une ligne de commandes.

```
bash-2.08$ ls > resultat

bash-2.08$ cat resultat
fichier
resultat

bash-2.08$ cat <<toto >>resultat
> une ligne en plus
> toto

bash-2.08$ cat resultat
fichier
resultat
une ligne en plus

bash-2.08$ ls toto 1>resultat 2>&1

bash-2.08$ cat resultat
ls: toto: No such file or directory
```

Sous le capot

UNIX est multitâches : il simule l'exécution simultanée de plusieurs processus grâce à un **ordonnanceur de tâches** (*scheduler*) qui choisit d'exécuter et de basculer d'un processus à un autre très rapidement.



➥ La commutation de tâches permet de simuler le parallélisme d'exécution des processus.

Représentation interne

Un processus est une zone mémoire de taille fixe qui permet de stocker :

- les informations sur le processus lui même
- le **code** : les instructions à exécuter (dans le langage du processeur)
- la **zone de données** : les variables manipulées par le code
- la **pile d'exécution** : les paramètres d'appels des fonctions

Un processus est donc représenté comme un programme qui s'exécute et qui possède son propre compteur ordinal (l'adresse en mémoire de la prochaine instruction à exécuter).

Les informations nécessaires au fonctionnement d'un processus (exécution, arrêt, reprise, etc.) constitue le **contexte d'exécution** de celui-ci.

Analogie classique

Soit un informaticien qui prépare un gâteau d'anniversaire pour sa fille.

Il a une recette pour faire le gâteau et dispose de farine, d'œufs, de sucre ...

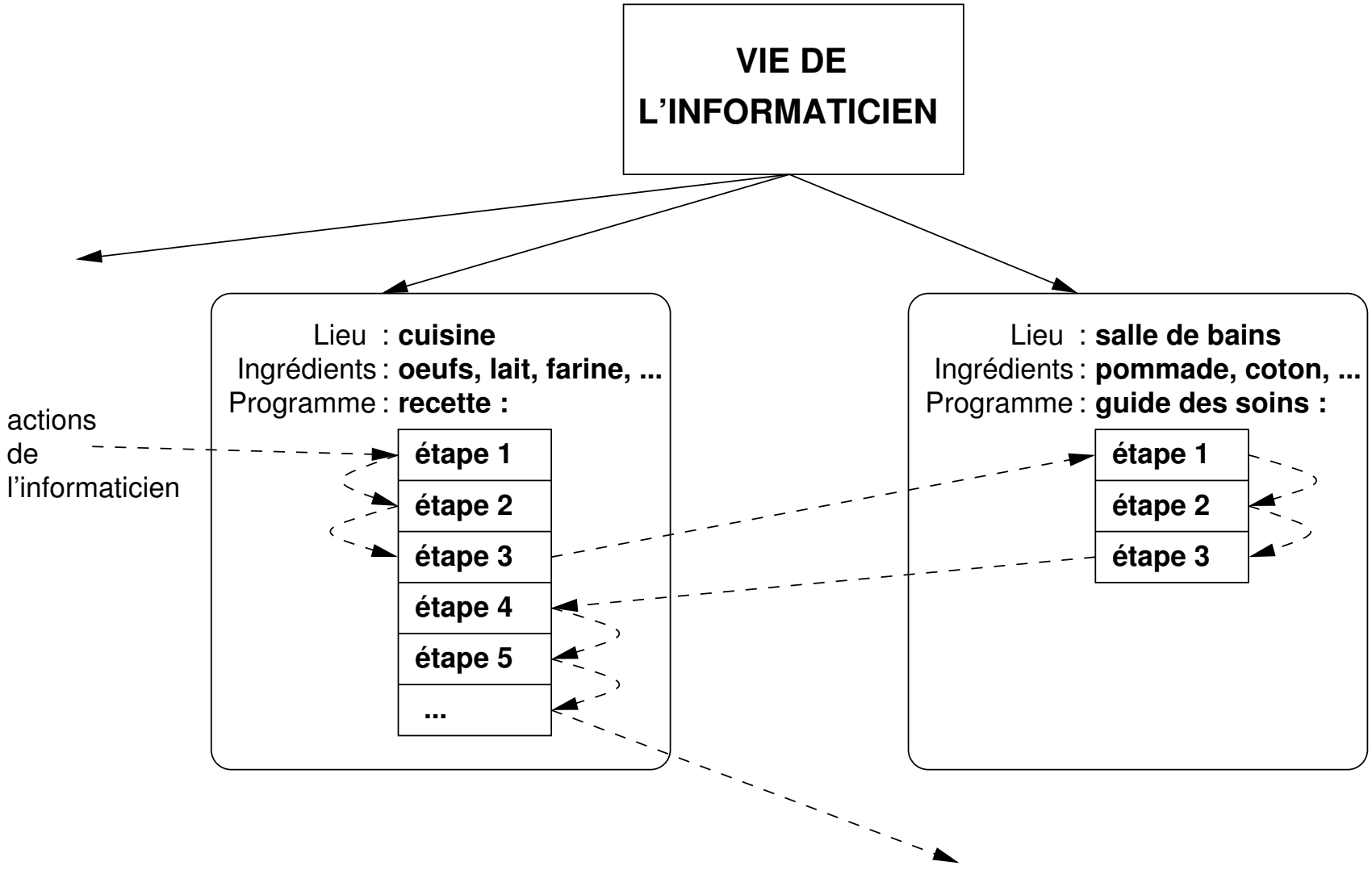
Ici la recette représente le programme (algorithme traduit en une suite d'instructions), l'informaticien joue le rôle du processeur (CPU) et les ingrédients sont les données à traiter.

Le processus est l'activité de notre cordon bleu qui lit la recette, trouve les ingrédients nécessaires et fait cuire le gâteau.

Si le fils de l'informaticien arrive en pleurant parce qu'il a été piqué par une guêpe, son père marque l'endroit où il était dans la recette (*l'état du processus en cours et son contexte sont sauvegardés*), cherche un livre sur les premiers soins et commence à soigner son fils.

Le processeur passe donc d'un processus (la cuisine) *à un autre plus prioritaire* (les soins médicaux), *chacun d'eux ayant un programme propre* (la recette et le livre des soins).

Lorsque la piqûre de la guêpe aura été soignée, l'informaticien reprendra sa recette à l'endroit où il l'avait abandonnée.


VIE DE
L'INFORMATICIEN
Lieu : cuisine
Ingrédients : oeufs, lait, farine, ...
Programme : recette :
étape 1
étape 2
étape 3
étape 4
étape 5
...
actions
de
l'informaticien
Lieu : salle de bains
Ingrédients : pommade, coton, ...
Programme : guide des soins :
étape 1
étape 2
étape 3

- à l'échelle d'une journée les actions de l'informaticien ont été faites en séquence (les unes après les autres)
- à l'échelle de l'année elles ont toutes été faites en même temps (le même jour !)

➥ pseudo parallèlisme

- Une même activité pourrait avoir été fait dans différents endroits (soins dans la cuisine plutôt que dans la salle de bains par exemple).
- Une même activité aurait pu être faite sur d'autres objets (œufs du voisins par exemple).

➥ notion de contexte d'exécution

Changements d'états