Cours « système d'exploitation »
1ère année
IUT de Caen, Département d'Informatique
Année 2000 − 2001
(François Bourdon)

Chapitre 3
Les bases du système (partie 2)

Plan

3.2 Processus et commandes associées

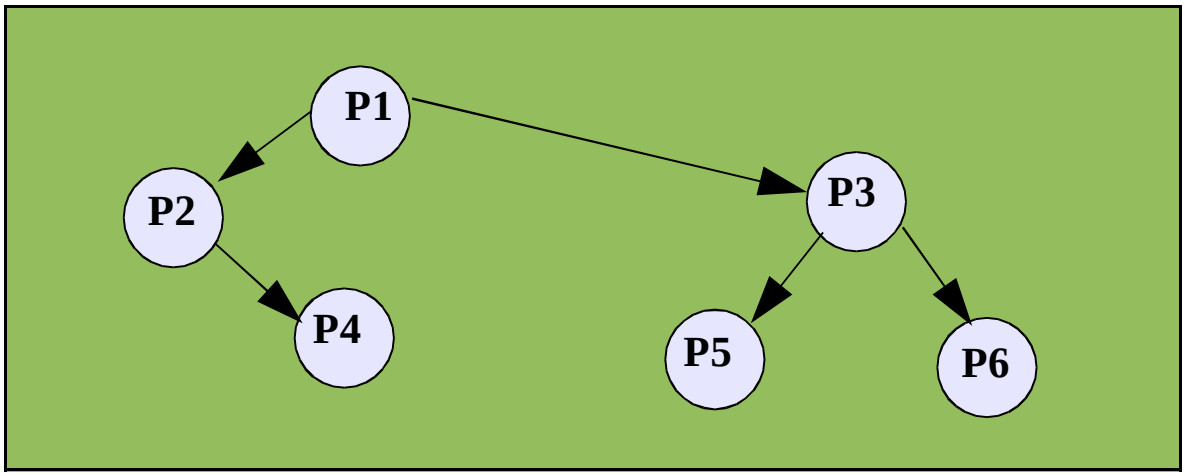
A. Définition

Les processus correspondent à l'exécution de tâches (programmes des utilisateurs, entrées−sorties, ...) par le système.

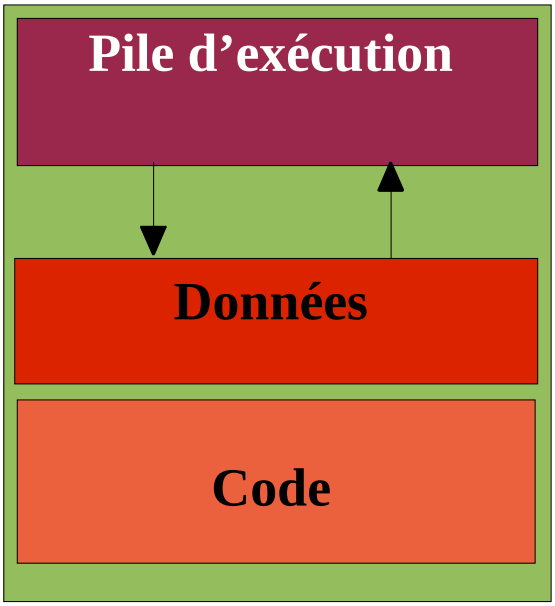
Un système d'exploitation doit en général traiter plusieurs tâches en même temps. Comme il n'a, la plupart du temps (UNIX), qu'un processeur, il résout ce problème grâce à un pseudo−parallélisme. Il traite une tâche à la fois, s'interrompt et passe à la suivante. La commutation de tâches étant très rapide, il donne l'illusion d'effectuer un traitement simultané.

Le mot ***processus*** désigne un programme binaire en cours d'exécution, c'est−à−dire un programme dans un environnement.

Les processus des utilisateurs sont lancés par un interprète de commande (shell). Ils peuvent eux−même lancer ensuite d'autres processus. On appelle le processus créateur, le père, et les processus créés, les fils. Les processus peuvent donc se structurer sous la forme d'une arborescence. Au lancement du système, il n'existe qu'un seul processus, qui est l'ancêtre de tous les autres.



Les processus sont composés d'un espace de travail en mémoire formé de 3 segments :



Le code correspond aux instructions, en langage d'assemblage, du programme à exécuter.

La zone de données contient les variables globales ou statiques du programme ainsi que les allocations dynamiques de mémoire.

Enfin, les appels de fonctions, avec leurs paramètres et leurs variables locales, viennent s'empiler sur la pile.

Les zones de pile et de données ont des frontières mobiles qui croissent en sens inverse lors de l'exécution du programme.

Un processus est donc un «programme» qui s'exécute et qui possède :

- son propre compteur ordinal (@ de la prochaine instruction exécutable),
- ses registres et
- ses variables.

Le concept de processus n'a donc de sens que dans le cadre d'un contexte d'exécution. Conceptuellement, chaque processus possède son propre processeur virtuel, en réalité, le vrai processeur commute entre plusieurs processus.

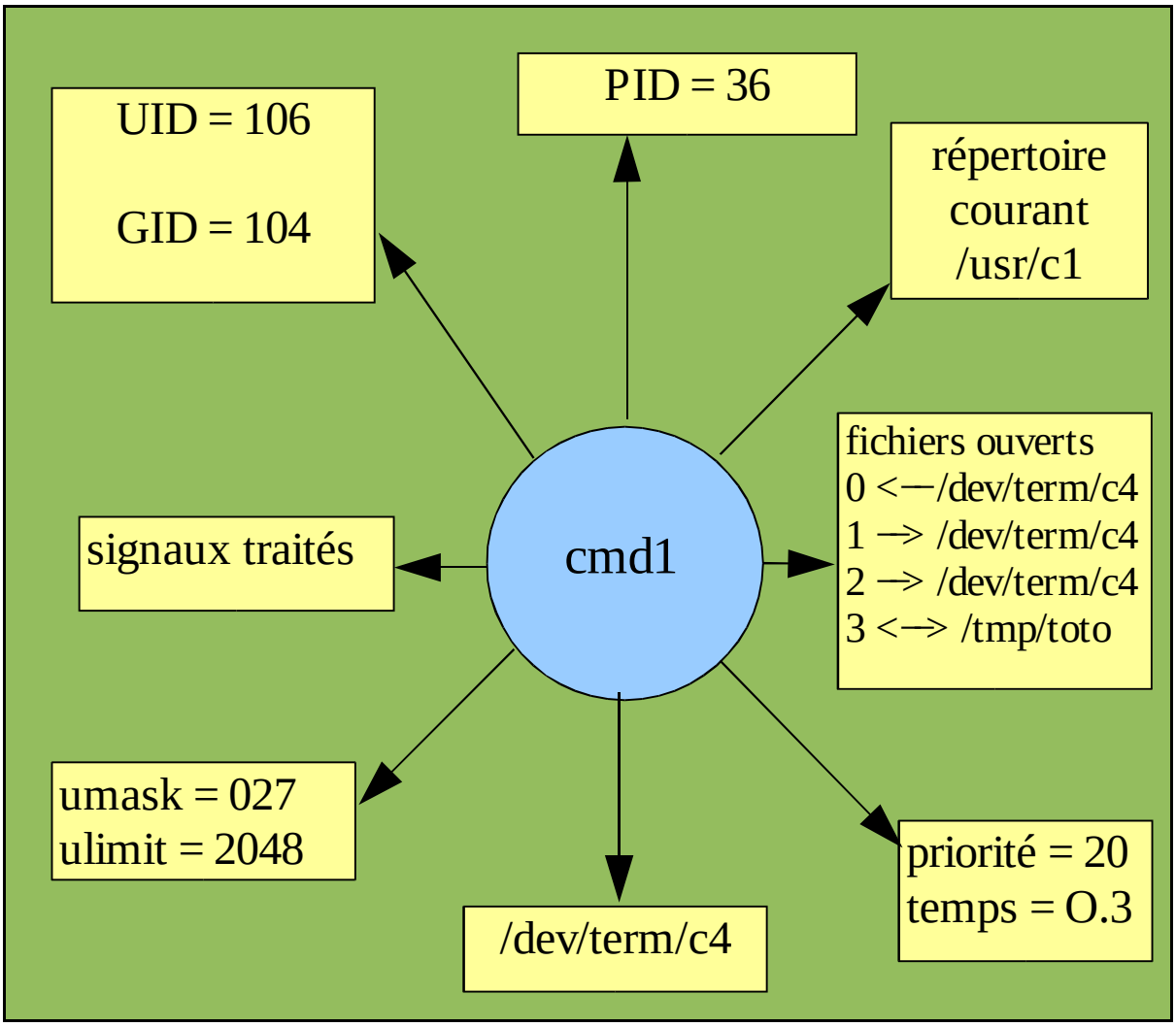
Le noyau maintient une table pour gérer l'ensemble des processus. Cette table contient la liste de tous les processus avec des informations concernant chaque processus.

Le nombre des emplacements dans cette table des processus est limité pour chaque système et pour chaque utilisateur.

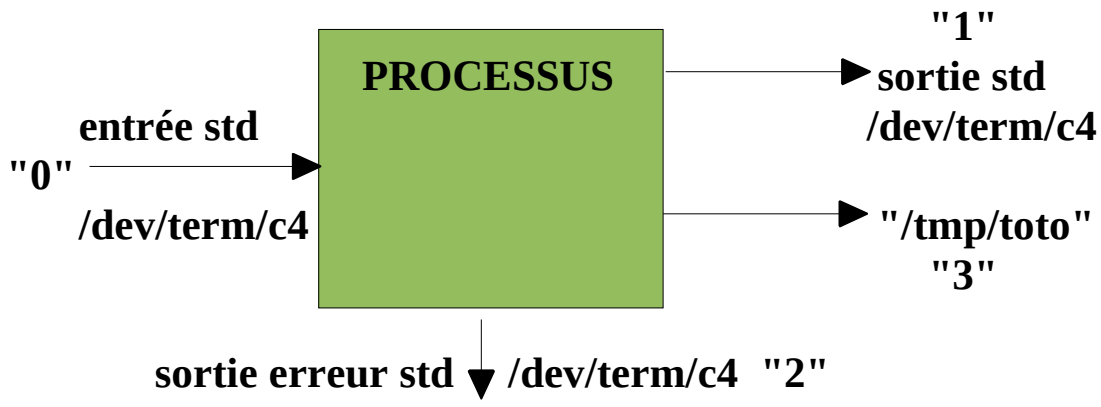
L'environnement d'un processus encore appelé ***contexte***, comprend entre autre :

- un numéro d'identification unique appelé **PID** (**P**rocess **ID**entifier) ;
- le numéro d'identification de l'utilisateur qui a lancé ce processus, appelé **UID** (**U**ser **ID**entifier), et le numéro du groupe auquel appartient cet utilisateur, appelé **GID** (**G**roup **ID**entifier) ;
- le **répertoire courant** ;
- les **fichiers ouverts** par ce processus ;
- le **masque** de création de fichier, appelé *umask* ;
- la **taille maximale** des fichiers que ce processus peut créer, appelée *ulimit* ;
- la **priorité** ;
- les **temps d'exécution** ;
- le ***terminal de contrôle***, c'est−à−dire le terminal à partir duquel la commande a été lancée.

Voici un premier schéma (simplifié) d'un processus :

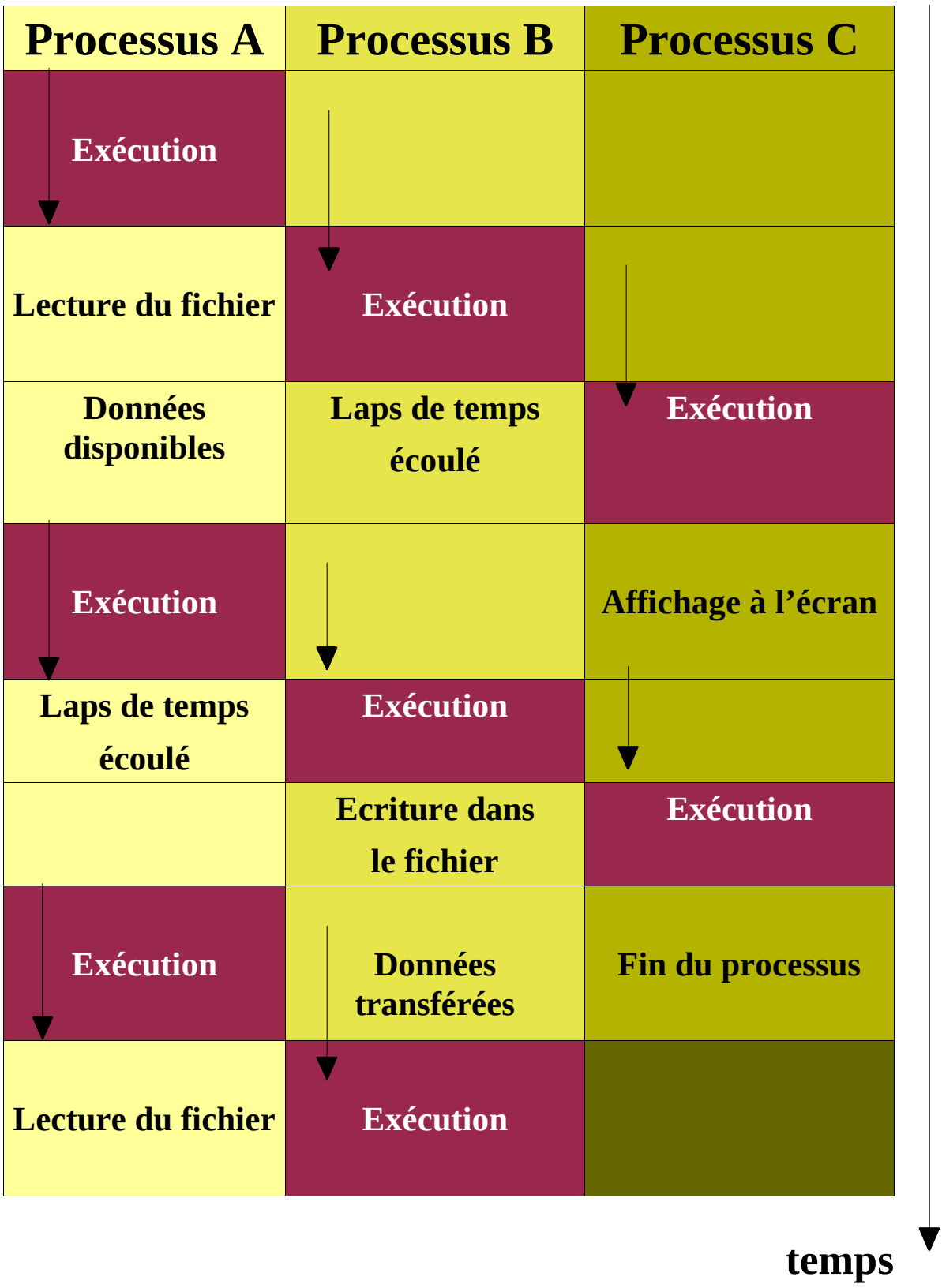


Ce prosessus a le numéro *36*. Il a été lancé par l'utilisateur qui a *106* pour UID. Il est en train d'exécuter le programme *'cmd1'*. Il a *consommé 0.3* seconde, avec une priorité de *20*. Son masque de création de fichier est *027*. Son terminal de contrôle est */dev/term/c4*. Son répertoire courant est */usr/c1*. Il a 4 fichiers ouverts :



Note : 0 = entrée std, 1 = sortie std, et 2 = sortie std d'erreur.

Sur le schéma ci−dessous, les trois programmes deviennent trois processus indépendants qui ont chacun leur propre contrôle de flux (compteur ordinal).



Sur un intervalle de temps assez grand, tous les processus ont progressé, mais à un instant donné un seul processus est actif.

La commutation des tâches (**multiprogrammation**), c'est−à−dire le passage d'une tâche à une autre, est réalisée par un ordonnanceur (scheduler) au niveau le plus bas du système. Cet ordonnanceur est activé par des interruptions d'horloge, de disque et de terminaux.

A chaque interruption correspond un vecteur d'interruption, c'est−à−dire un emplacement mémoire contenant une adresse. L'arrivée de l'interruption provoque le branchement à cette adresse.

Comme le processeur commute entre les processus, la vitesse d'exécution d'un processus ne sera pas uniforme et variera vraisemblablement si les mêmes processus étaient exécutés à nouveau.

Il ne faut donc pas que les processus fassent une quelconque présomption sur le facteur temps.

Différence entre un processus et un programme :

Un **processus** est une activité d'un certain type qui possède un **programme**, des données en entrée et en sortie, ainsi qu'un état courant.

Un seul processeur peut être partagé entre plusieurs processus en se servant d'un algorithme d'ordonnancement qui détermine quand il faut suspendre un processus pour en servir un autre.

Une métaphore illustrant la différence entre processus et programme :

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'oeufs, 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 fournir.

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 est sauvegardé), 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.

B. Commandes

Certaines des caractéristiques de l'environnement peuvent être consultées par diverses commandes. Nous connaissons déjà :

pwd	affiche le chemin du répertoire courant
tty	affiche le terminal de contrôle
umask	affiche le masque de création de fichier

Voici d'autres commandes :

id	consulte l'UID et le GID.

Exemple :

```
$ id
uid=106(c1)          gid=104(cours)
$
```

logname	affiche uniquement le nom associé à l'UID.

Exemple :

```
$ logname
c1
$
```

Le PID est stocké dans une pseudo−variable spéciale que l'on appelle **$.**

On peut consulter le PID du shell courant en tapant :

```
$ echo $$
36
$
```

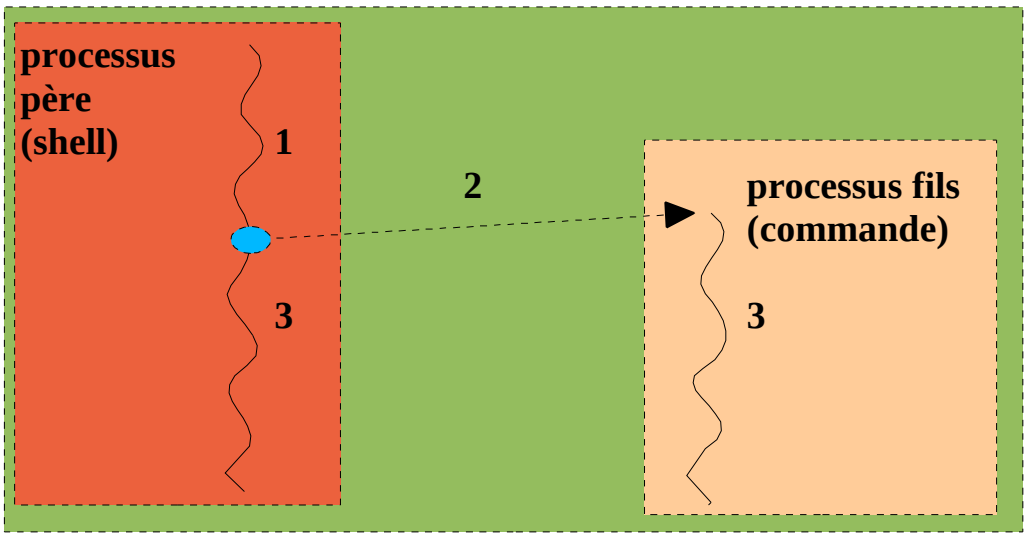
Le 1er "**$**" définit le contenu de la pseudo−variable, alors que le second "**$**" correspond au PID du shell courant.

C. Création de processus

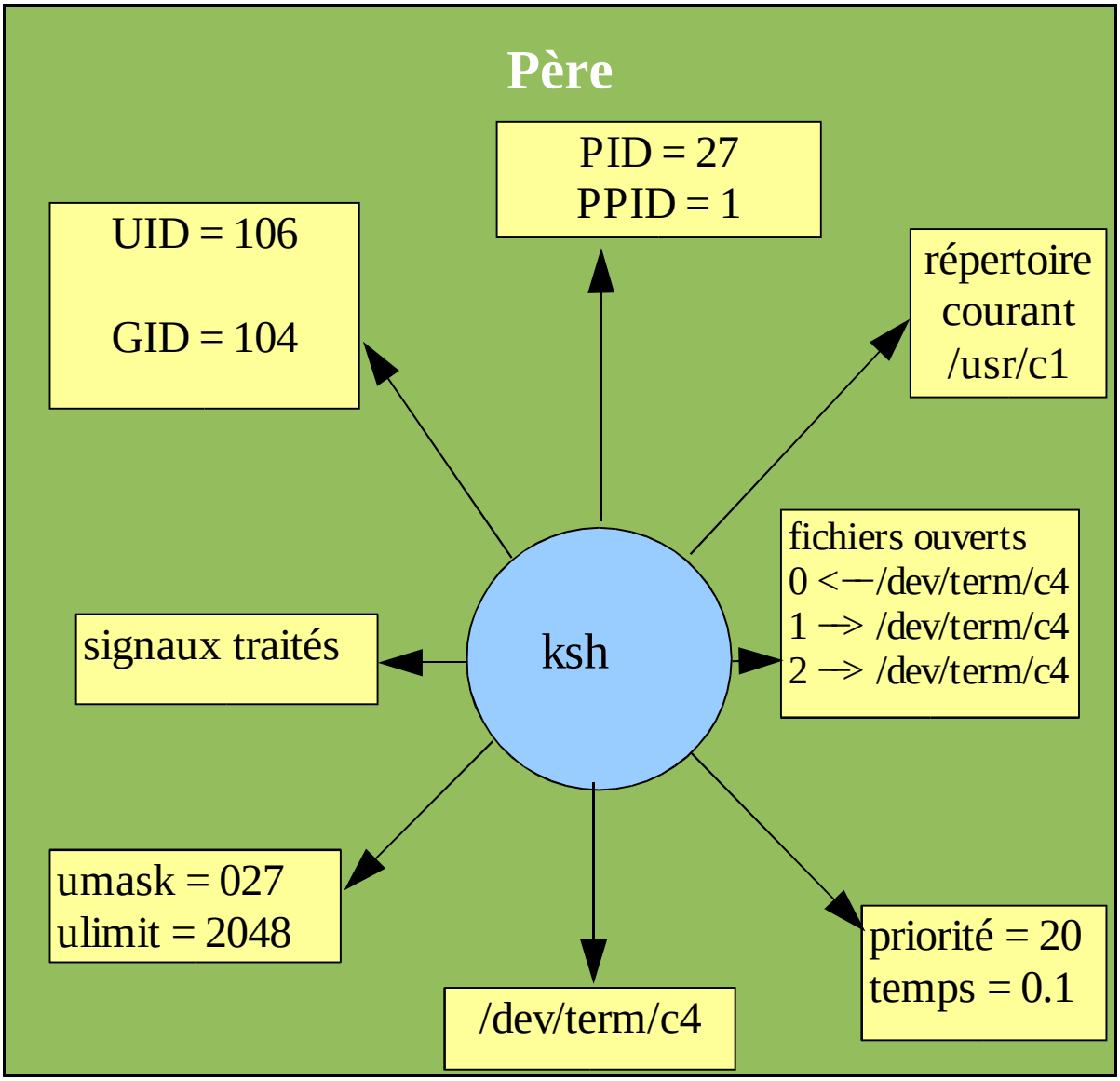
- Pour chaque commande lancée (sauf les commandes internes), le shell crée automatiquement un nouveau processus.

- Il y a donc 2 processus. Le premier, appelé **processus *père***, exécute le programme shell, et le deuxième, appelé **processus *fils***, exécute la commande.

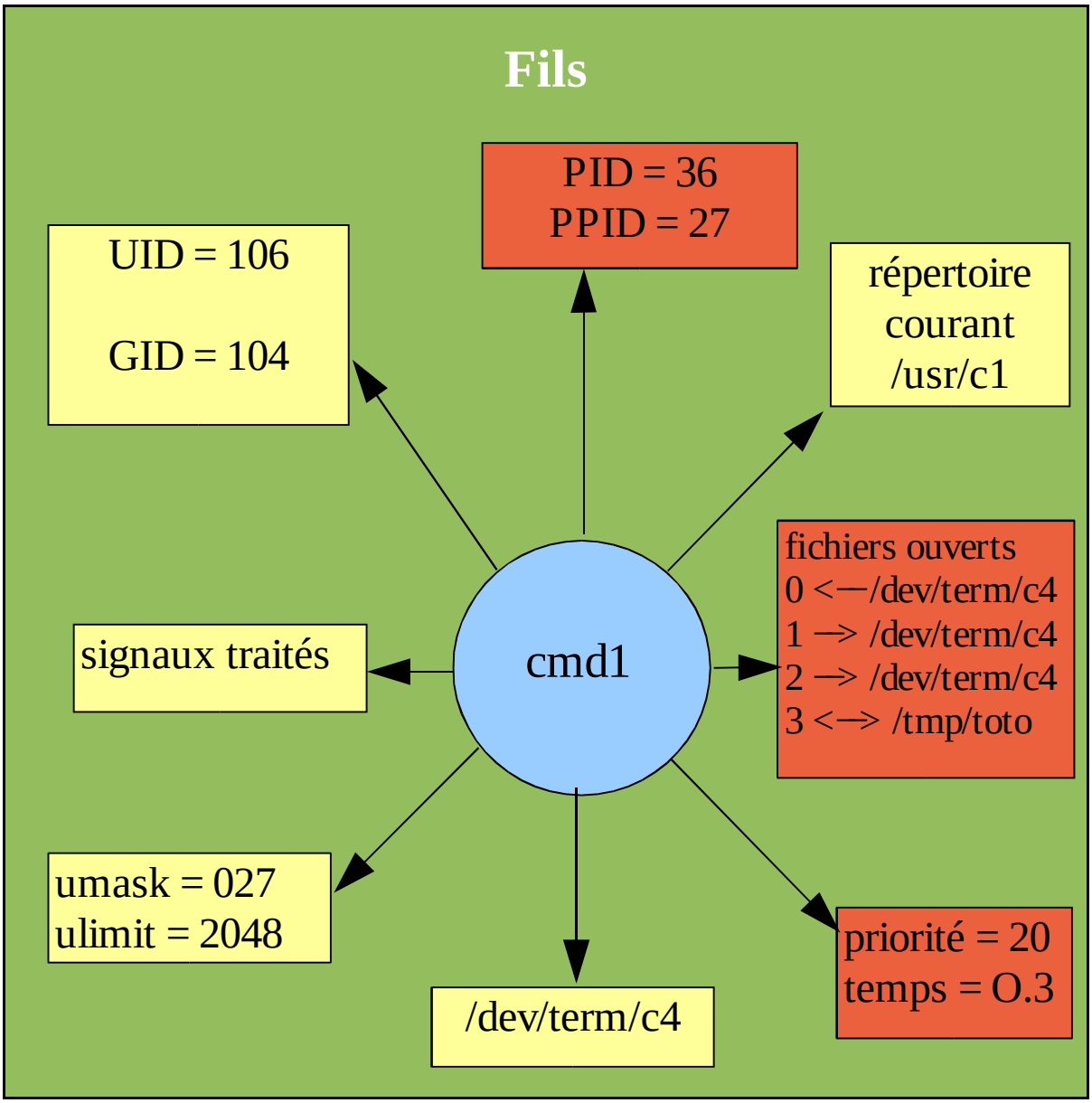
- Le fils ***hérite*** de tout l'environnement du père, sauf bien sûr du **PID**, du **PPID** et des temps d'exécution.



Un nouvel élément de l'environnement apparaît ici, le ***PPID***. C'est le **PID** du processus père. Le père du processus 36 est le processus 27, et celui de 27 est le processus 1. Seul le fils (36) a ouvert le fichier **/tmp/toto**.

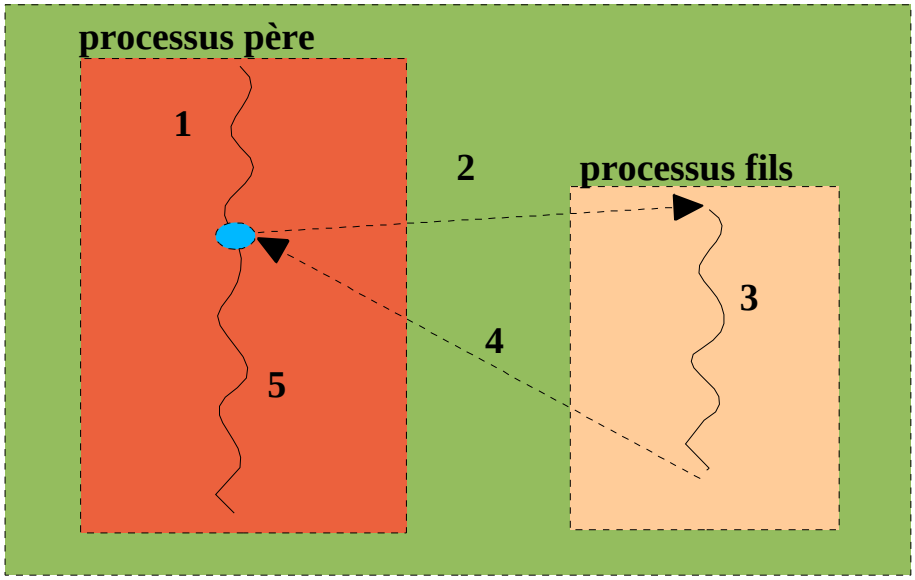


Les deux processus peuvent parfaitement tourner *en parallèle*. La puissance de traitement est partagée entre tous les programmes lancés et, mis à part les machines multi−processeurs, un seul processus est actif à un instant *t*.



On utilisera cette solution (processus lancés en parallèle) par exemple pour lancer un traitement très long, et continuer à travailler en même temps. Dans ce cas, on dit que le père a lancé un fils en ***tâche de fond (background)*** ou encore en ***mode asynchrone***.

Une autre solution consiste à placer le processus père en attente jusqu'à ce que le processus fils soit terminé.



Pour lancer une commande en plaçant le père en attente, il suffit de taper cette commande :

```
$ cmd1
... résultat de la commande cmd1
$
```

Ce mode est donc le mode par défaut dans le shell.

Pour lancer une commande en tâche de fond, il faut faire suivre cette commande par le caractère '**&**':

```
$ cmd1 &
[1]  127
$
```

Le Korn Shell affiche un numéro de tâche entre [] et le PID de cette tâche de fond, puis continue à travailler, donc affiche la chaîne d'invite et attend la prochaine commande.

En Bourne Shell, il n'y a pas de numéro de tâche, la même commande aurait donnée :

```
$ cmd1 &
127
$
```

Si vous avez plusieurs commandes successives à lancer en arrière plan, il faudra utiliser les parenthèses.

```
$ (cmd1; cmd2) &
[2]  128
$
```

La commande **cmd2** ne sera lancée que lorsque la commande **cmd1** sera terminée. Ceci dit, l'utilisateur récupère la main tout de suite. Le shell détecte la présence du '**&**' partout sur la ligne.

Dans le cas suivant, la commande **cmd1** sera lancée par un sous−processus shell :

```
$ (cmd1)
$
```

La commande **cmd1** est lancée en arrière plan et la commande **cmd2** est tout de suite lancée derrière en direct.

```
$ cmd1 & cmd2
[3]  130
$
```

La commande **wait** *n* permet d'attendre la mort de la tâche de fond dont le PID est *n*.

```
$ cmd1 &
[4]  132
$ wait 132
... on est bloqué jusqu'à ce que cmd1 se termine
$
```

Si *n* n'est pas précisée, **wait** attend la mort de toutes les tâches de fond. **wait** ne s'applique qu'aux processus lancés dans le shell lui−même.

D. Arborescence de processus

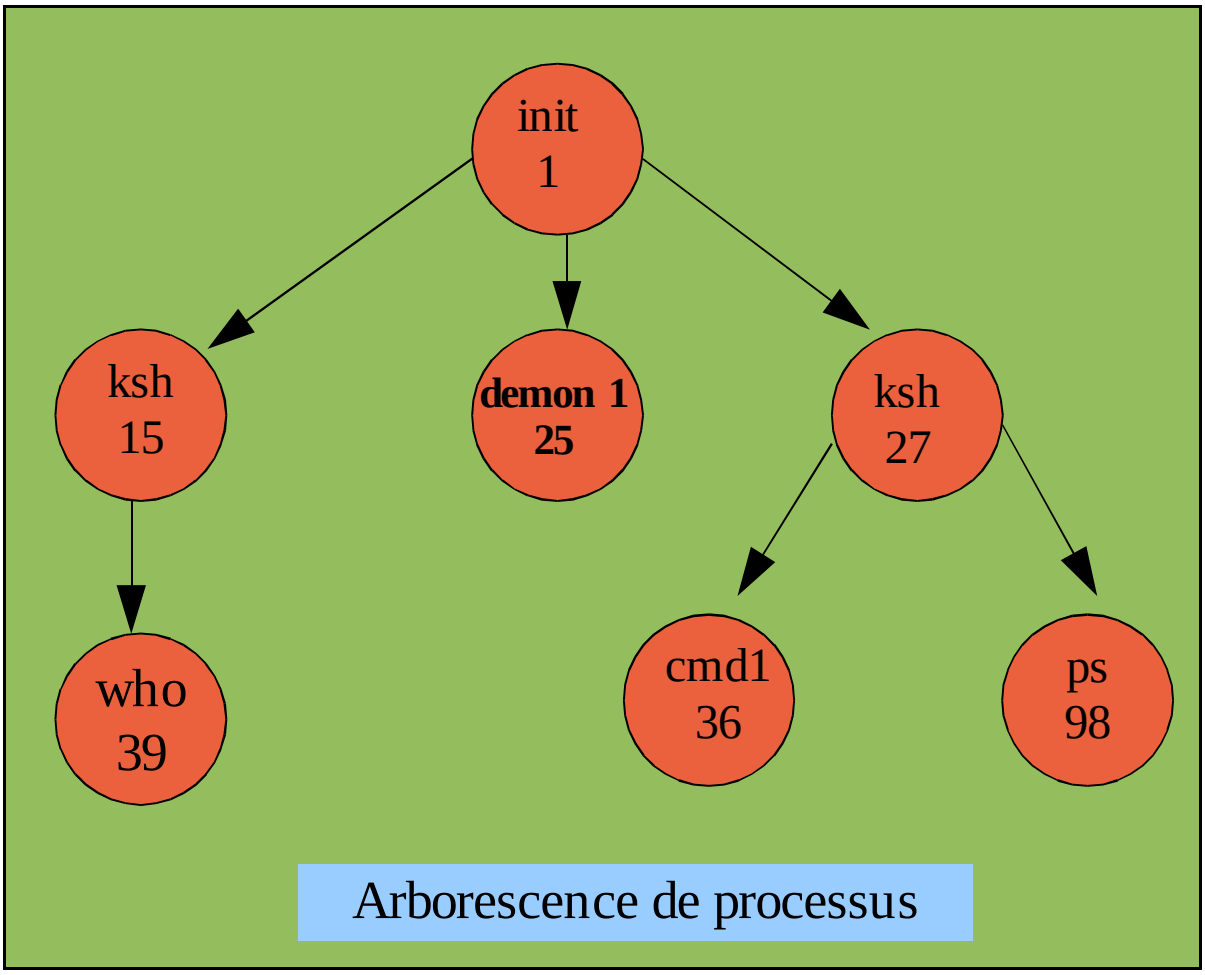
Tous les processus sont créés à partir d'un processus père, existant déjà.

> Le premier processus est un peu spécial. Il est créé lorsque le système est initialisé. Il s'appelle "***init***", a le **PID *1*** et n'est associé à aucun terminal. Son travail consiste à créer de nouveaux processus.

Le processus "***init***" crée 2 sortes de processus :

- des **démons**, c'est−à−dire des processus qui ne sont rattachés à aucun terminal, qui sont **endormis** la plupart du temps, mais qui se **réveillent** de temps en temps pour effectuer une tâche précise (par exemple la gestion des imprimantes).

- des **processus interactifs**, associés aux lignes d'entrées/sorties sur lesquelles sont rattachés des terminaux. Autrement dit des processus vous permettant de vous connecter.



Pour visualiser les processus que vous avez lancé, tapez la commande «**ps**» :

```
$ ps
PID        TTY          TIME       COMMAND
527        ttyp4        1:70       −ksh
536        ttyp4        0:30       cmd1
559        ttyp4        0:00       ps
$
```

- **PID** identifie le processus,
- **TTY** est le numéro du terminal associé,
- **TIME** est le temps cumulé d'exécution du processus,
- **COMMAND** est le nom du fichier correspondant au programme exécuté par le processus.

Dans cet exemple un utilisateur s'est connecté au système sur le terminal *ttyp4* (commande *tty*). Un processus a donc été créé pour exécuter le programme shell (*ksh*). Il a le PID *527*.

```
$ echo $$
527
$ cmd1 &
$
$ ps
PID        TTY          TIME      COMMAND
527        ttyp4        1:70      −ksh
536        ttyp4        0:30      cmd1
559        ttyp4        0:00      ps
$
```

Puis l'utilisateur a lancé le programme **cmd1** en tâche de fond. Un deuxième processus (de PID *536*) est créé qui exécute la commande '***cmd1***'.

Le processus *527* continue à tourner en parallèle, l'utilisateur peut donc continuer à travailler, et il lance la commande **ps**. Un troisième processus (de PID *559*) est créé pour exécuter cette commande. La commande ps affiche le résultat sur la sortie standard, celle−ci étant associée à l'écran, le résultat apparaît.

Pendant ce temps, le shell est endormi. Lorsque la commande ps est terminée, le processus *559* meurt, son père, le processus *527*, est averti par le système, et il recommence à travailler. Comme il s'agit du shell, il affiche à nouveau la chaîne d'invite '**$**' et attend une nouvelle commande.

Sans option, la commande concerne les processus associés au terminal depuis lequel elle est lancée.

E. Identificateurs réels et effectifs

A chaque processus sont en fait associés deux groupes d'identifications :

1. l'UID et le GID *réels* identifient l'utilisateur qui a lancé le processus.

2. l'UID et le GID *effectifs* (EUID et EGID), identifient les droits des processus.

Le plus souvent, les identités réelles et effectives sont identiques. Cependant, il peut être nécessaire de modifier les droits d'un processus.

Le fichier **/etc/passwd** est protégé en écriture. Aucun utilisateur n'a le droit d'écrire dans ce fichier. Cependant, en utilisant la commande **passwd** (*/bin/passwd*), vous écrivez quand même dans ce fichier, puisque vous modifiez votre mot de passe.

```
$ ls −l /etc/passwd
−rw−r−−r−− 1 root sys 7100 Aug 22 15:21 /etc/passwd
$ type passwd
passwd is /bin/passwd
$ ls −l /bin/passwd
−r−sr−sr−x 1 root sys 19040 Dec 19 1985 /bin/passwd
$
```

Remarquez que pour le fichier exécutable **/bin/passwd**, **ls** affiche un caractère '*s*' à la place du caractère '*x*' habituel, montrant par là que ce fichier a le mode « *set user ID* » et « *set group ID* ».

Dans ce cas, l'EUID et l'EGID du processus sont ceux du propriétaire du fichier exécuté (et non pas ceux de l'utilisateur qui l'a lancé).

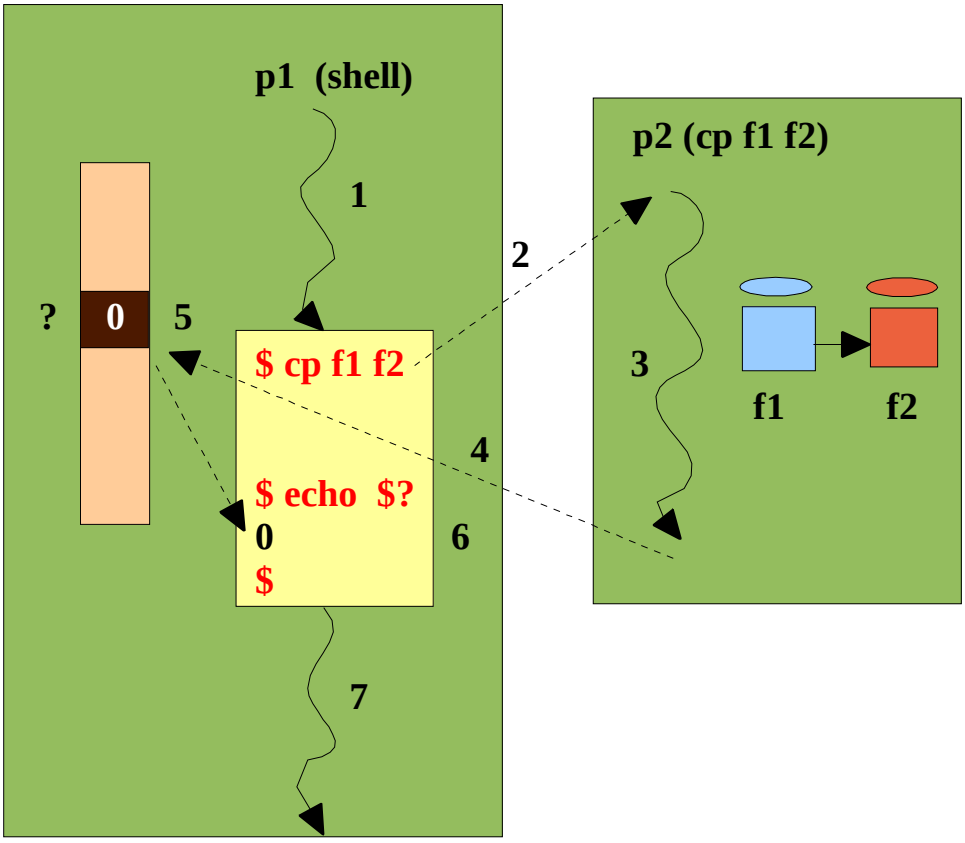
Ainsi quiconque lance l'exécutable **passwd**, travaille avec l'UID effectif de *root* et GID de *sys*.

F. Statut

Lorsqu'un processus se termine, il retourne toujours une ***valeur significative*** ou ***statut***.

> Par convention, lorsqu'un processus se termine correctement, il retourne la valeur 0, sinon il retourne une valeur différente de 0 (généralement 1). Ce choix permet de ramener des codes significatifs pour différencier les erreurs.

Le statut d'une commande shell est placé dans la pseudo−variable spéciale, nommée "?". On peut consulter sa valeur en tapant la commande :



Sur cet exemple, la valeur **0** renvoyée par la commande "**echo**" nous indique que la commande "**cd**" s'est bien passée.