Système d'Exploitation (SE)
Mémoire et Processus

Dr. Mohammed BOUGAA
mbougaa@itescia.fr

Plan du cours

- SE et Gestion des Processus
- SE et Gestion de la Mémoire

Partie 1:

Gestion des Processus

Processus

Un processus est un programme **en cours d'exécution**

- Programme (= instructions machine) : entité statique
- Processus (= réalisation d'actions) : entité dynamique

A un **processus** sont associes plusieurs éléments :

- son code
- ses données
- les ressources qui lui sont attribuées
- un ou plusieurs threads d'exécution

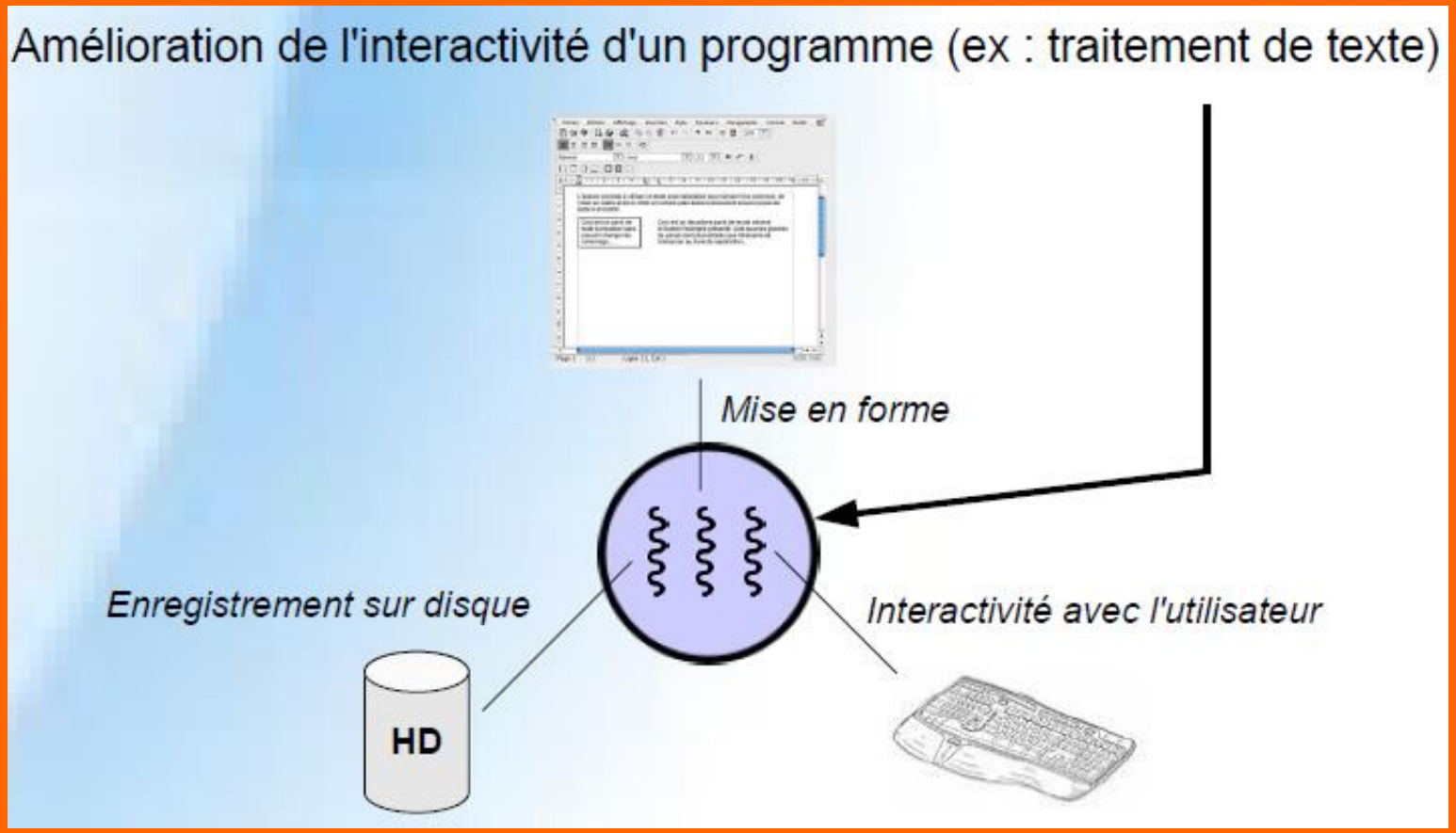
Thread

Un **thread** est un flot d'exécution dans le code du processus doté :

- d'un compteur programme (suivi des instructions à exécuter)
- de registres systèmes (variables de travail en cours)
- d'une pile (historique de l'exécution)

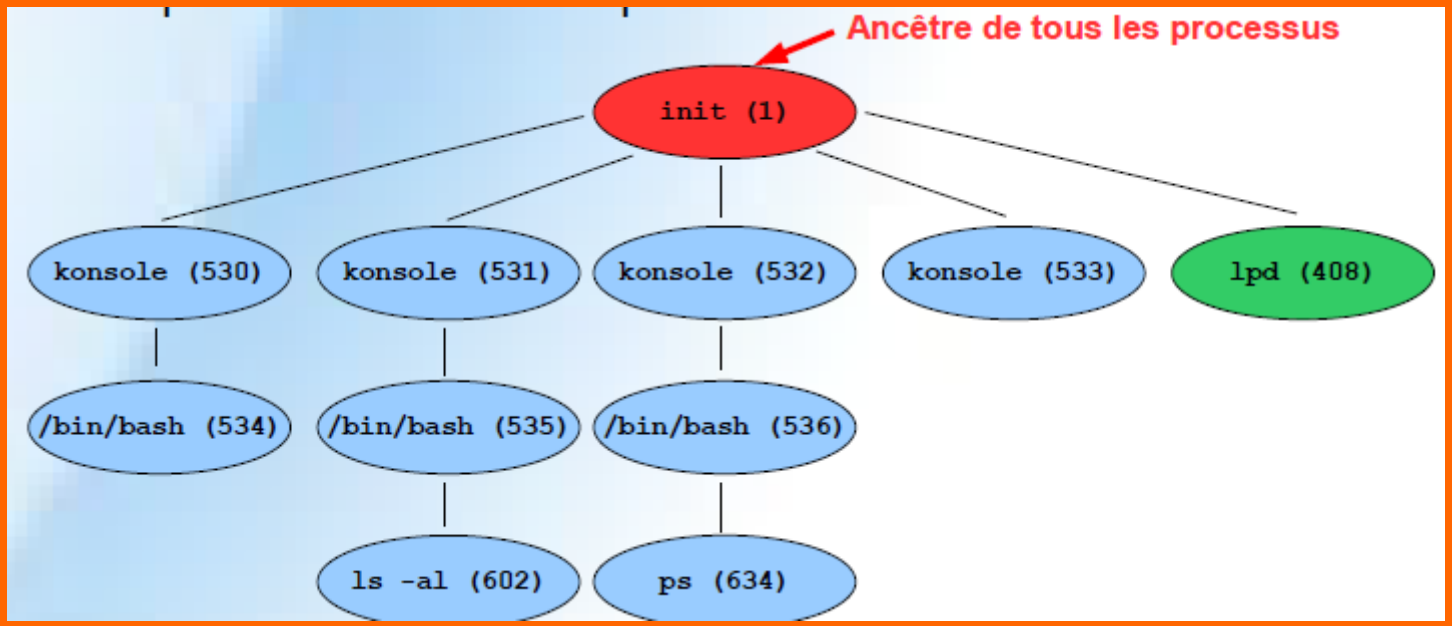
- Plusieurs processus permettent a un ordinateur d'effectuer plusieurs taches a la fois. Ils se partagent les **ressources physiques**.
- Plusieurs threads permettent a un processus de décomposer le travail a exécuter en parallèle. Ils se partagent les **ressources physiques et virtuelles**.

Utilisation des thread



- Alors qu'un **processus** sert à lancer deux fois le même éditeur de texte par exemple.

Exemple d'arborescence de processus :



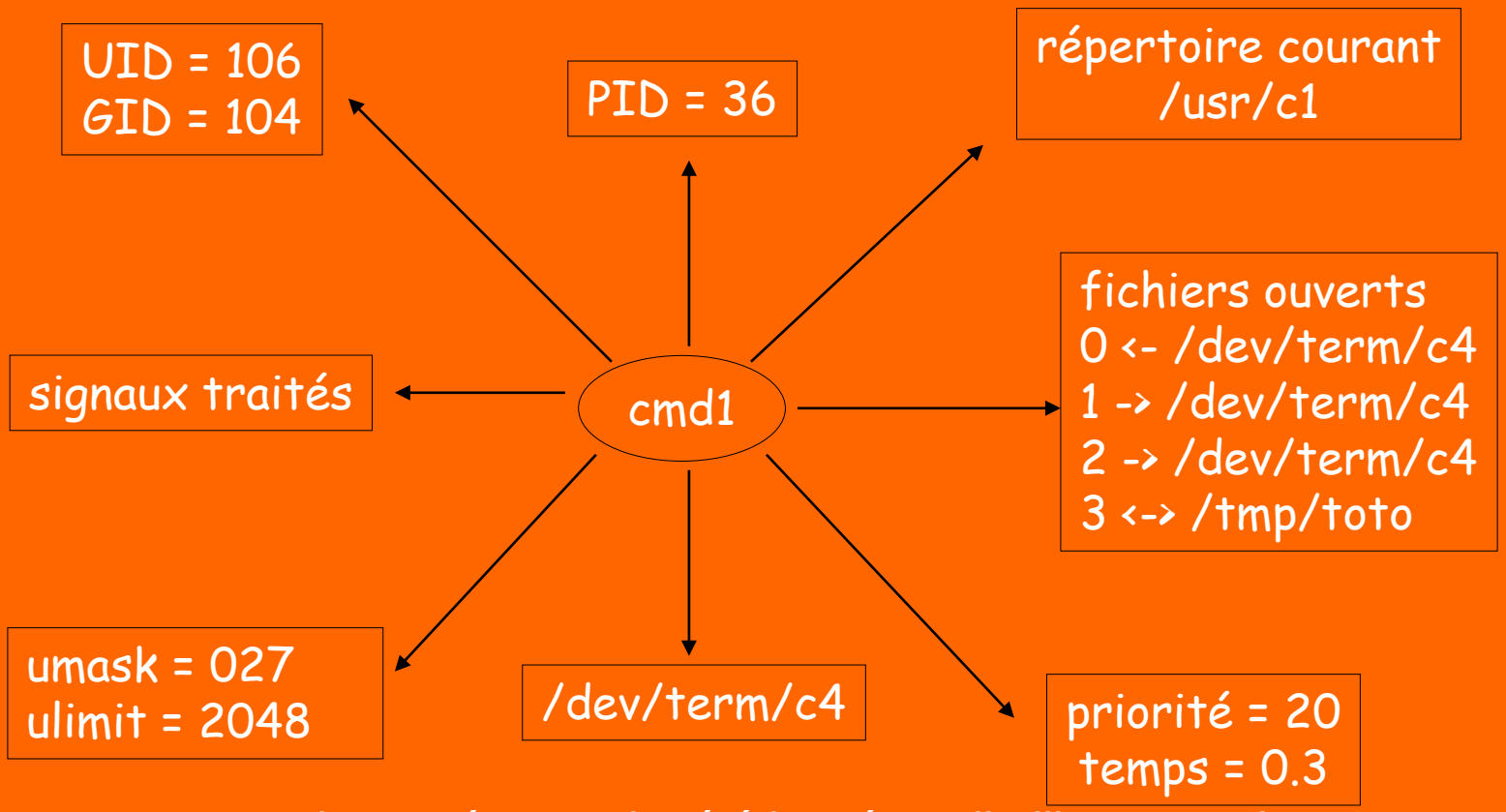
En **Bleu** ce sont des **Processus utilisateur** et en **Vert** ce sont des **Processus système** (démons)

Un processus est créé par un autre processus (**init** est le père de tout les processus)

L’environnement d'un processus UNIX

- le programme qu'il exécute (**CMD**)
- un numéro d'identification que lui affecte le système (**PID**)
- un créateur (**PPID**)
- l'utilisateur pour lequel il fonctionne (**UID**)
- le terminal ou la fenêtre du processus (**TTY**)
- une consommation CPU (**CPU**)
- une consommation mémoire (**MEM**)
- une durée de traitement (**TIME**)
- une heure de lancement (**STIME**)
- un facteur de priorité (**C**)...
- les fichiers ouverts par ce processus
- . . .

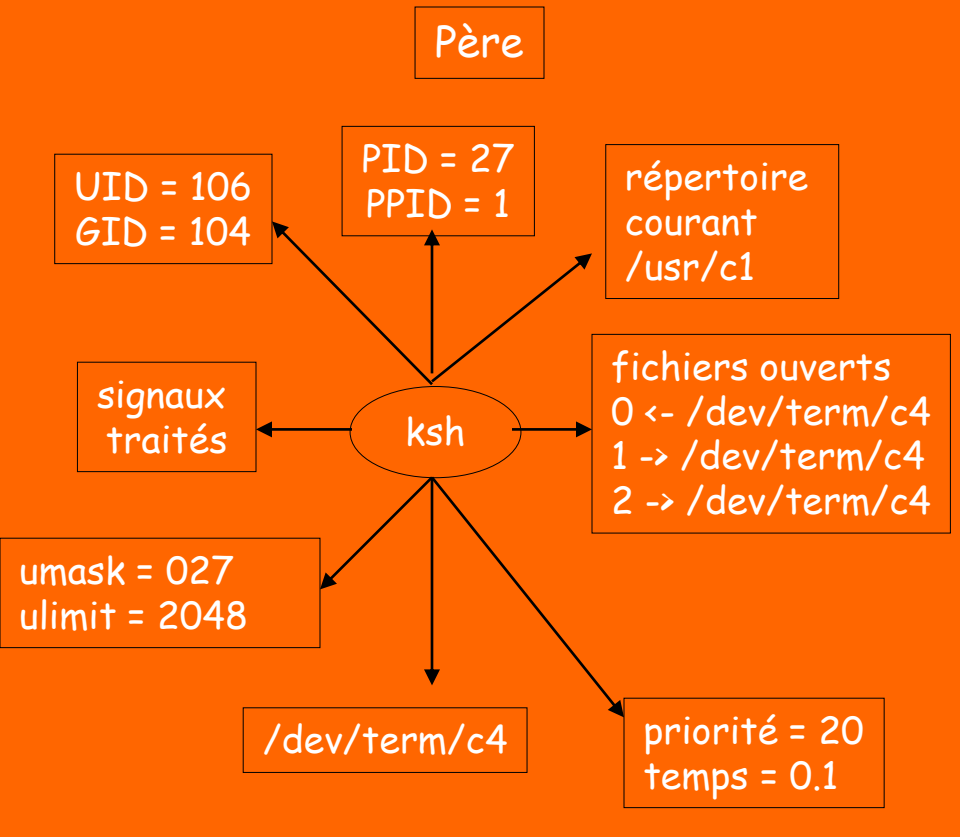
Un exemple : schéma d'un processus Unix

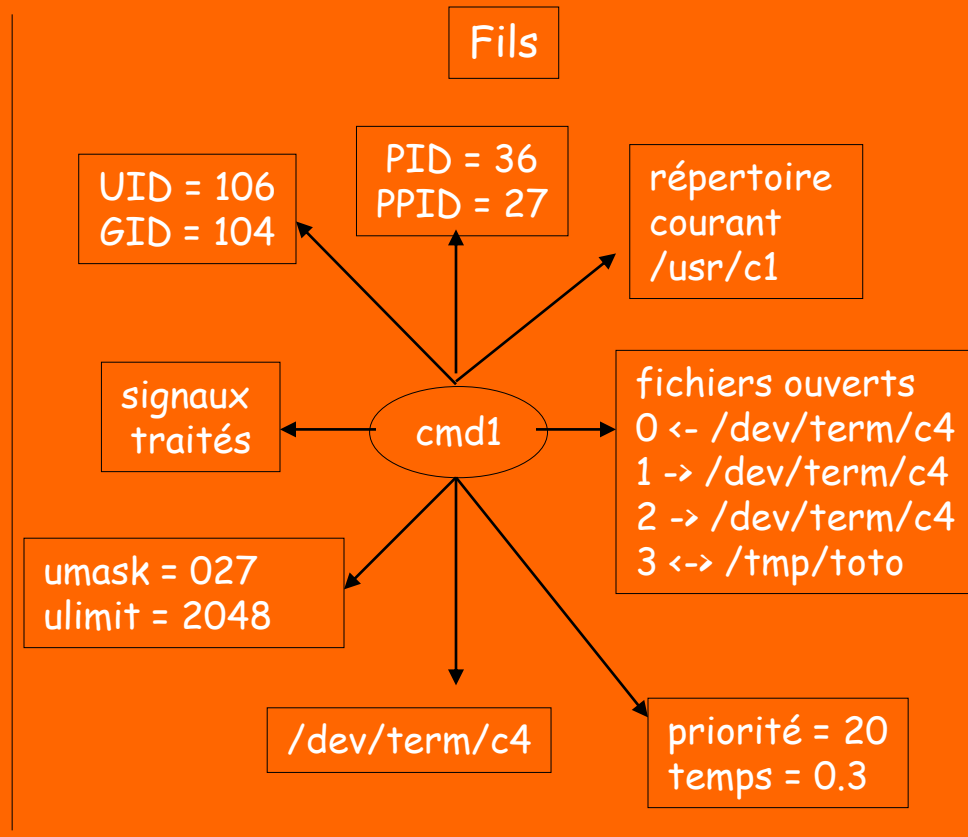


Ce processus a le numéro 36. Il a été lancé par l'utilisateur qui a 106 pour UID. Il est entrain d'exécuter le programme '**cmd1**'. Il a consommé **0.3** seconde, avec une priorité de **20**. Son masque de création est 027. Son terminal de contrôle est **/dev/term/c4**. Son répertoire courant est **/usr/c1**. Il a 4 fichiers ouverts : **0, 1, 2, et 3**.

Structure d'un processus Unix

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





Commandes UNIX de gestion des processus

La commande ps

```
ps [-options]
```

- -a : affiche des renseignements sur tous les processus attachés à un terminal
- -l : donne, pour chaque processus, le nom de l'utilisateur (user), le pourcentage de cpu (%cpu), la taille totale du processus dans la mémoire (size), la mémoire réservée (rss) en Ko …
- -x : affiche également des informations sur les processus non liés au terminal
- ….

Commandes UNIX de gestion des processus

Commande ps –Exemple-:

```
% ps
  PID    TTY      STAT     TIME        CMD
  746    pts/3    S        00:00:00    -bash
  749    pts/3    S        00:00:02    gs
  848    pts/3    S        00:03:28    mozilla-bin
  965    pts/3    S        00:00:00    ps
```

- PID : le numéro d'identification du processus
- TTY : le terminal depuis lequel le processus a été lancé
- STAT : l'état du processus au moment du lancement de la commande
  - R : le processus est en cours d'exécution
  - T : le processus est stoppé
  - S : le processus dort depuis moins de 20 secondes
  - I : le processus dort depuis plus de 20 secondes
  - Z : le processus en attente d'un message du noyau
- TIME : le temps d 'exécution de la commande
- CMD : le libellé de la commande lancée

Commandes UNIX de gestion des processus

La commande nice

Changement de la priorité d'un processus

```
nice -valeur commande
nice -5 gcc programme.c]
```

La commande kill

Destruction d'un processus

```
kill -9 no_processus
kill -9 521
```

La commande &

Lancement en arrière-plan d'un processus

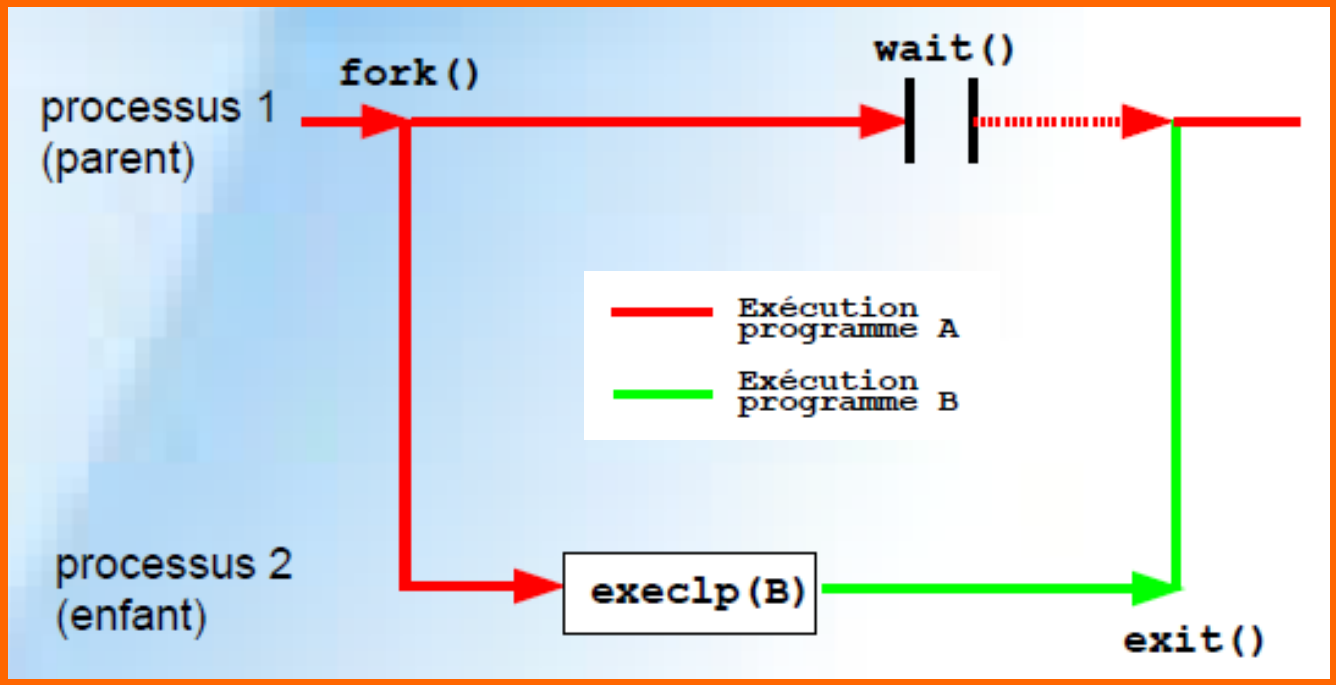
```
nom_processus&
prog1&
```

La commande jobs

Affichage des processus en background

```
jobs
```

Manipulation de processus en langage C -Principes-



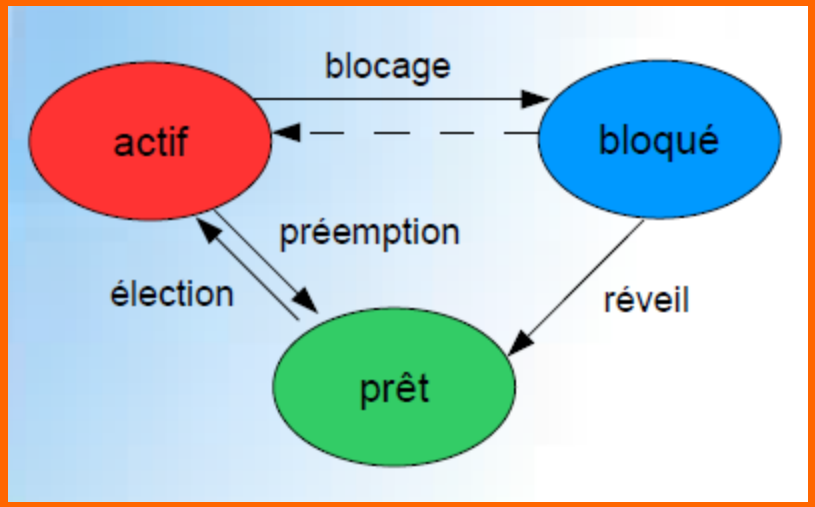
Manipulation de processus en langage C -Principes-

Fonctions primitives de gestion de processus en C

- **int fork ();**
- **int execlp (char **comm*, char **arg0*, ..., NULL);**
- **void exit (int *status*);**
- **int wait (int **ptr_status*);**
- **pid_t waitpid(pid_t *pid*, int **status*, int *opts*)**
- **int sleep (int *seconds*);**
- **int getpid ();**
- **int getppid ();**

Différents états d'un processus

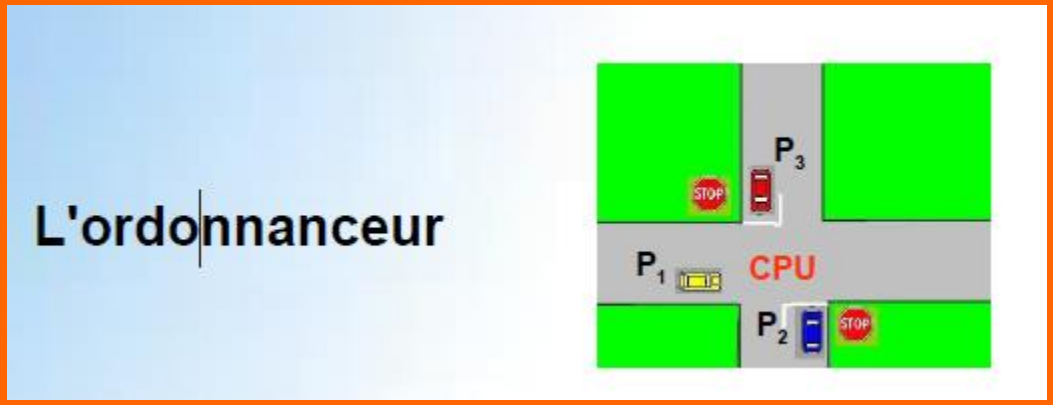
Les processus peuvent être dans plusieurs états, les plus courants étant :



actif : le processus s'exécute sur le processeur
prêt : le processus est prêt a s'exécuter mais n'a pas le processeur
bloqué : il manque au processus une ressource (en plus du processeur) pour qu'il puisse s'exécuter

Allocation du processeur aux processus

Le noyau gère l'ordonnancement de tous les processus du système
Les processus peuvent être dans plusieurs états



L'ordonnanceur est un composant (procédure) du noyau du système d'exploitation

L'ordonnancement consiste a **choisir** le processus a exécuter a un instant *t* et à **déterminer** le temps durant lequel le processeur lui sera alloué et d'optimiser certains aspects des performances du système

Inter-Process Communication (IPC) : méthodes permettant à plusieurs processus de communiquer entre eux

3 categories de mécanismes :

Outils permettant aux processus de s'**echanger des donnees**

- les fichiers
- la memoire partagee

Outils permettant de **synchroniser des processus**

- les semaphores
- les signaux

Outils permettant d'**echanger des donnees et de synchroniser des processus**

- les tubes
- les files d'attente de messages

Exemple de création de processus

Exemple

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  pid_t pid;
  pid = fork();
  switch(pid) {
  case 0 :
    printf("Je suis le fils : mon PID est %d
            et mon PPID est %d\n",getpid(),getppid());
  break;
  case -1 :
    perror("Erreur de creation de processus avec fork");
  break;
  default :
    printf("Je suis le pere : mon PID est %d
            et mon PPID est %d\n",getpid(),getppid());
  break;
 }
 return 0;
}
```

Plus d'infos sur fork(): http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

Exemple de synchronisation entre processus

Synchronisation

- `exit(i))`
  - Termine un processus, i est un octet (donc valeurs possibles : 0 à 255) renvoyé dans une variable du type `int` au processus père.
- `pid_t wait(int *status)`
  - Suspend le processus jusqu'à la terminaison de l'un de ses fils
  - Retourne le pid du fils, ou -1 dans le cas où il n'y a pas (ou plus) de fils
  - Achèvement du père : fils pris en charge par `init`
  - `status` : valeur de `exit` ou autre (dans le cas d'un signal)
  - `waitpid` pour attendre un processus particulier dont on connaît son `pid`.

Exemple de synchronisation entre processus

Exemple

```c
#include <unistd.h>     /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <stdio.h>      /* Input/Output */
#include <sys/wait.h>  /* Wait for Process Termination */
#include <stdlib.h>     /* General Utilities */

int main() {
   int retval; int status;
   pid_t p = fork();
   if (p == 0) {
     sleep(1); /* sleep for 1 second */
     printf("CHILD: Enter an exit value (0 to 255): ");
     scanf("%d", &retval);
     exit(retval);
   } else if (p > 0) {
       printf("PARENT: I will wait for my child to exit.\n");
       wait(&status);
       printf("PARENT: Child exit code is: %d\n",
           WEXITSTATUS(status));
       exit(0);
   } else { exit(-1); }
}
```