FONCTIONS D'UN ENVIRONNEMENT D'EXÉCUTION DESTINÉ À FOURNIR UNE MACHINE VIRTUELLE À L'UTILISATEUR.

FONCTION DE GESTION DE L'INFORMATION

- **structuration des programmes, des données, des objets de l'utilisateur,**
- **conservation,**
- **désignation (espace d'adressage, fichiers, objets),**
- **contrôle d'accès (droits d'accès, listes de capacités) aux objets utilisés ou mis en commun**

FONCTION DE COMMUNICATION

- **avec les autres machines virtuelles locales ou distantes sur d'autres sites,**
- **transferts en entrées-sorties,**
- **accès à distance de procédure, de services, de fichiers, d'objets**
- **mobilité des objets ou des environnements, diffusion, réseau**

FONCTION D'EXÉCUTION ("contrôle")

- **permettre et suivre l'exécution de programmes en séquence, en parallèle, en concurrence,**
- **composition de programmes (chaînages par filtres, enchaînements, sous-systèmes, découpage, etc...).**

SERVICES DIVERS POUR L'ENVIRONNEMENT D'EXÉCUTION

- **aides à la mise au point,**
- **traitement des défaillances, reprises,**
- **mesure du temps, etc...**

MACHINE VIRTUELLE AU BON NIVEAU D'ABSTRACTION.

NIVEAU D'UNE APPLICATION PARTICULIÈRE

• c'est le rôle du logiciel d'application spécifique et de son IHM (interface homme machine)

NIVEAU FICHIERS

• langage de commande : tous les objets de la machine virtuelle sont des fichiers : fichier de stockage (txt, .doc, .obj, .bin, etc...), fichier exécutable, fichier périphérique, fichier autres sites.
• Interpréteur du langage de commande : exemple Shell Unix, MacOs

NIVEAU LANGAGE MACHINE

• Machine virtuelle (exemple : système Linux ou système Unix)
• Modèle machine Von Neumann : Adressage machine selon jeu d'instruction
• Conventions d'utilisation de l'espace d'adressage (ex. Linux ou Unix, Windows NT, AS400, VAX,...).
• Création d'un exécutable et évolution d'une image en espace d'adressage (mémoire virtuelle).
• Liaison statique, liaison dynamique.
• Processus (Linux, Unix, Posix), threads, gestion des processus vu de l'utilisateur, état, process_id,
• Autres aspects machine : mode maître, interruptions, entrées-sorties

NIVEAU LANGAGE ÉVOLUÉ

• C (en relation avec le modèle Posix), C++, Ada, Java, Modula, CSP,...

EXEMPLES D'AUTRES NIVEAUX

• objets locaux ou répartis, composants : plate-forme Corba, plate-forme Java, plate-forme Dcom

INTRODUCTION À L'ENVIRONNEMENT UTILISATEUR FOURNI PAR

les systèmes de la famille Posix, Unix, Linux

- notion de processus
- chaîne de production de programme et préparation de processus
- machine virtuelle
- espace d'adressage = mémoire virtuelle du processus
- primitives de création et de gestion de processus
- environnement multiprocessus
- ordonnancement des processus
- test et représentation de l'environnement d'un utilisateur

NOTION DE PROCESSUS

•• instance d'un programme mis sous une forme exécutable en mémoire

•• déroulement de l'exécution de cette instance par la machine virtuelle

• un processus peut se dérouler dans l'un des modes de l'architecture machine : en mode utilisateur ou en mode privilégié (maître, superviseur, noyau, système,..) •

•• objet actif qui modifie son état et ses données en utilisant la machine virtuelle ••

• un processus est caractérisé par son état, son contexte et son espace d'adressage •

MODÈLE DE PROGRAMME SOURCE

```
main (int argc, char *argv[], char *env[])
    /*   argc : nombre de paramètres de la fonction main(), nom du programme inclu */
    /*   *argv : tableau de pointeurs de caractères contenant le nom et la liste des paramètres */
    /*        argv[0] -> "nom de la fonction   */
    /*        argv[1] -> "1er argument         */
    /*   *env : tableau de pointeurs de caractères donnant accès à l'environnement du processus
{
    /*     corps du programme                  */
}
```

Exemple : une commande (d'un "shell") est un programme catalogué.

Soit la commande de fichier : copier fichier1 fichier2

argc vaut 3 car il y a 3 chaînes de caractères sur la ligne de commande

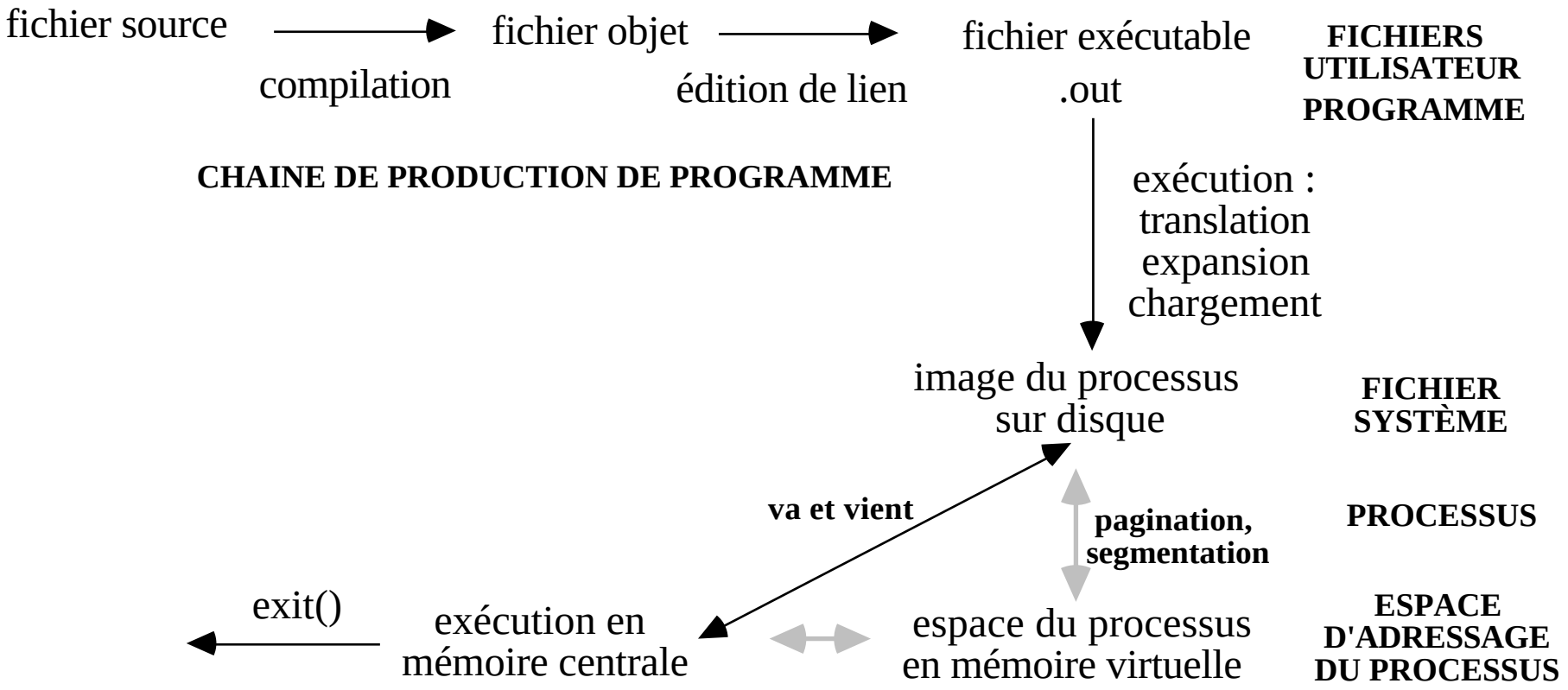
argv[0] : copier

argv[1] : fichier1

argv[2] : fichier2

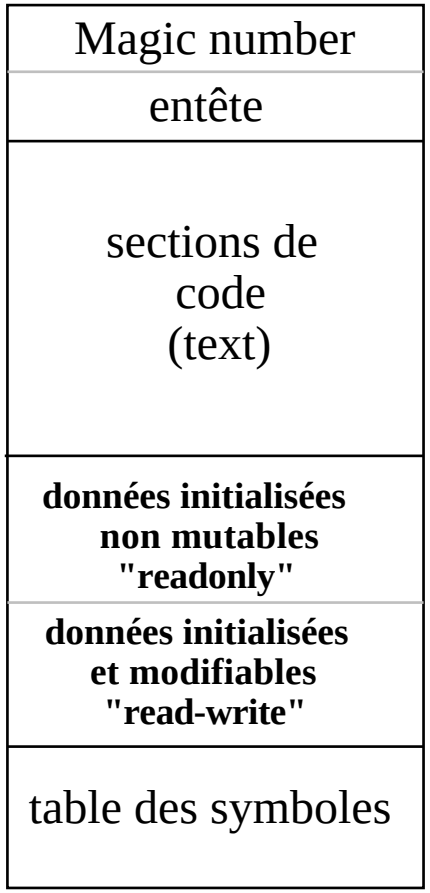
GÉNÉRATION D'UN EXÉCUTABLE ET UTILISATION D'UN EXÉCUTABLE

CRÉATION D'UN PROCESSUS ET EXÉCUTION D'UN PROCESSUS



REPRÉSENTATION EN FICHIER D'UN PROGRAMME EXÉCUTABLE

structure décrite par a.out.h :



Magic Number :

si #! : fichier exécutable par un interpréteur

#! est alors suivi par le chemin d'accès à l'interpréteur :

#!/bin/sh pour le Bourne Shell

#!/bin/csh pour le C-Shell

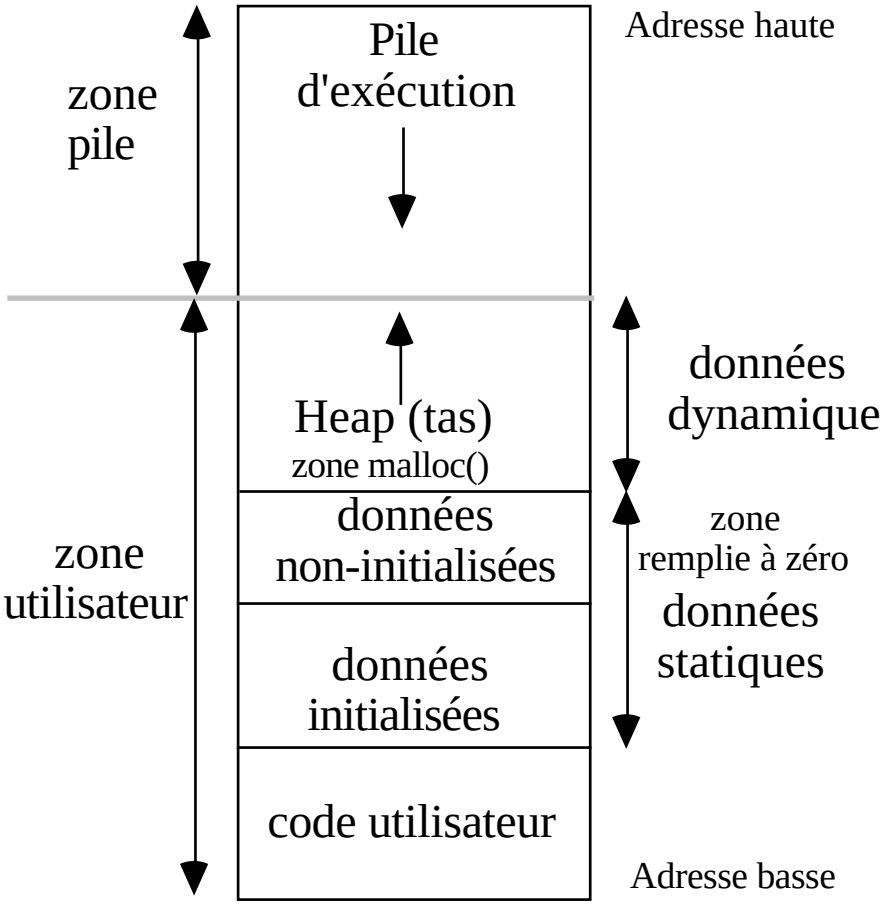
sinon le fichier est directement exécutable: c'est le résultat d'une compilation et d'une édition de lien. Ce champ indique :

• si la pagination est autorisée (stickybit),

• si le code est partageable par plusieurs processus (code réentrant).

Structuration analogue pour le format ELF dans Linux

Image d'un processus en mémoire virtuelle



•Conventions d'adressage:

répartition des adresses des objets de l'image du processus dans son espace d'adressage ou mémoire virtuelle.

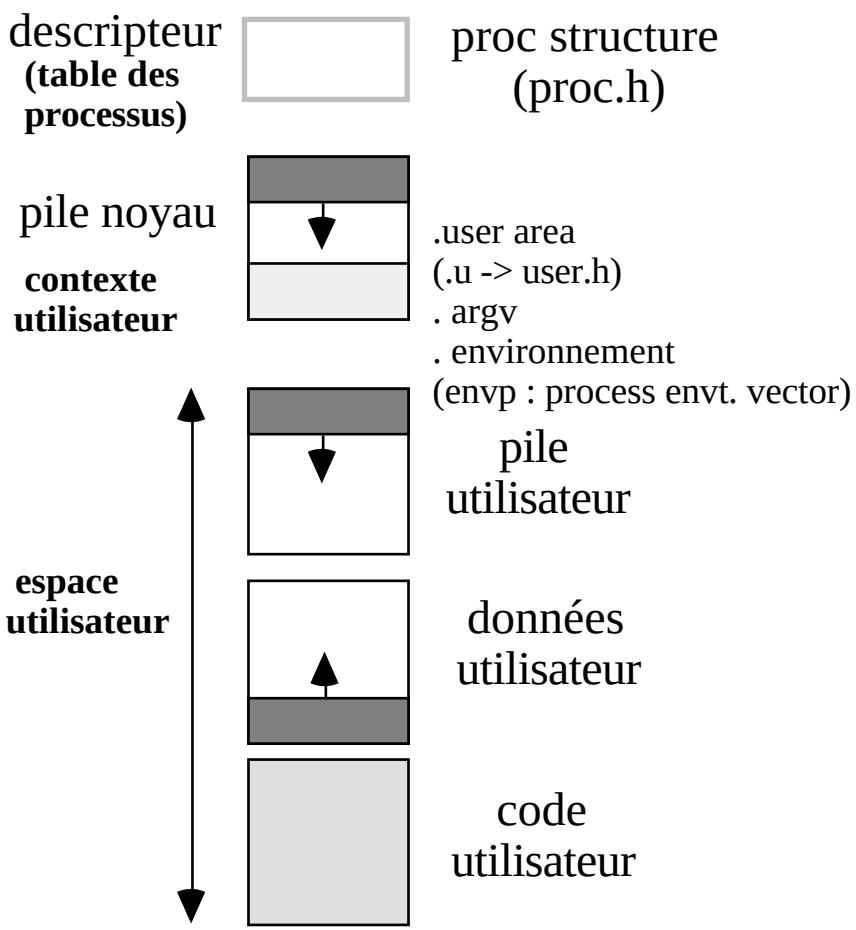
•Chargement en mémoire centrale des valeurs des objets de l'image du processus

Il peut se faire :

* soit au démarrage du processus,
* soit à la demande,

au fur et à mesure des défauts de pages (système avec pagination à la demande).

ESPACE D'ADRESSAGE DU PROCESSUS VU DU SYSTÈME



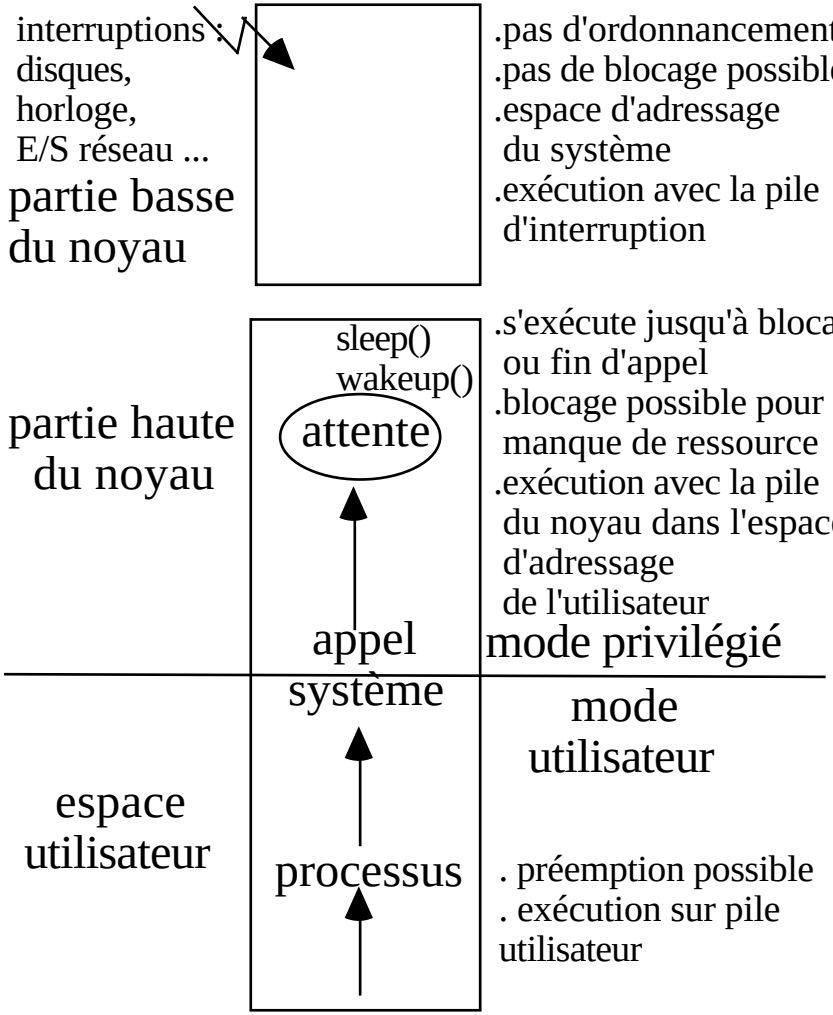
- Pile noyau utilisée quand le processus s'exécute en mode système,

un appel système provoque le passage du processus du mode d'exécution utilisateur au mode système, le processus exécute alors du code système pour son propre compte.

- "user area" : ensemble d'informations sur le processus,

informations nécessaires à l'ordonnancement des processus, même quand le processus n'est plus résident en mémoire physique (son image est alors sur le disque de va et vient pour pagination)

Modes d'exécution



DESCRIPTEUR DE PROCESSUS
- proc structure proc.h -

exemple de quelques attributs d'un processus

• infos d' ordonnancement
p_pri => priorité courante du processus
p_userpri => priorité du processus en mode utilisateur

• identificateurs
p_pid => identificateur du processus (unique sur la machine), s'obtient par la primitive **getpid()**
p_ppid => identificateur du créateur du processus, s'obtient par la primitive **getppid()**
p_uid => identificateur de l'utilisateur (user id) s'obtient par la primitive **getuid()**

• gestion mémoire
p_textp => pointeur vers un fichier exécutable
p_p0pbr => adresse de la table des pages du processus
p_szpt => taille de la table des pages
p_addr => localisation de la zone u. (user area)

• gestion d'événements et signaux
p_wchan => signaux attendus par le processus
p_sig **=>** masque des signaux en attente

• gestion du compte-temps
p_time => temps restant avant fin délai de garde

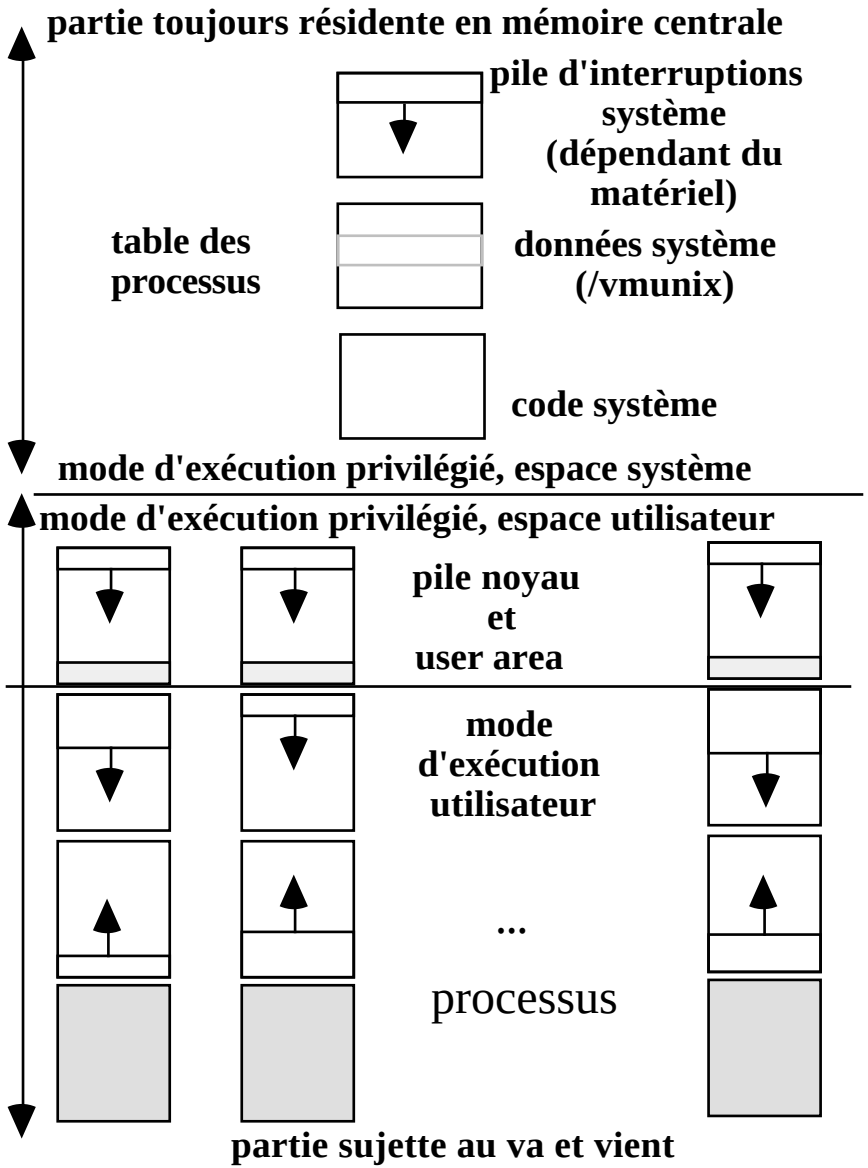
Structuration système

- **Espaces d'adressage des processus**
  pas forcément résident en mémoire centrale

  environnement programme
  mode d'exécution utilisateur
  espace utilisateur
  environnement système pour l'utilisateur
  mode d'exécution privilégié
  espace utilisateur

- **Espace système commun à tous les processus**
  doit être résident en mémoire centrale

  environnement système pour la communauté
  mode d'exécution privilégié
  espace système



Machine virtuelle utilisateur

1 utilisateur = plusieurs processus

• Tout processus peut créer un autre processus

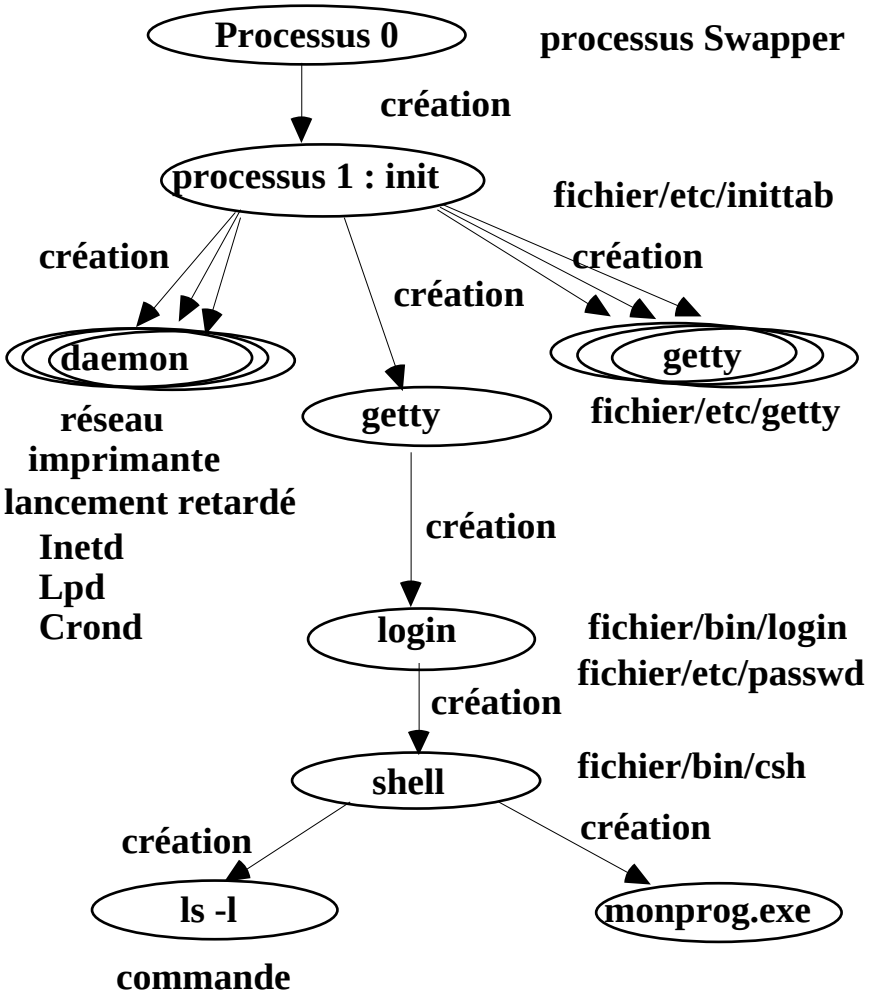
• Arborescence de processus avec un rapport père-fils entre créateur et créé

• Tout le système repose sur ce concept arborescent, y compris les processus permanents du système (démons ou "daemons")

création de processus par des appels système

fork() crée un nouveau processus

exec() charge un nouvel exécutable



Primitive : int fork

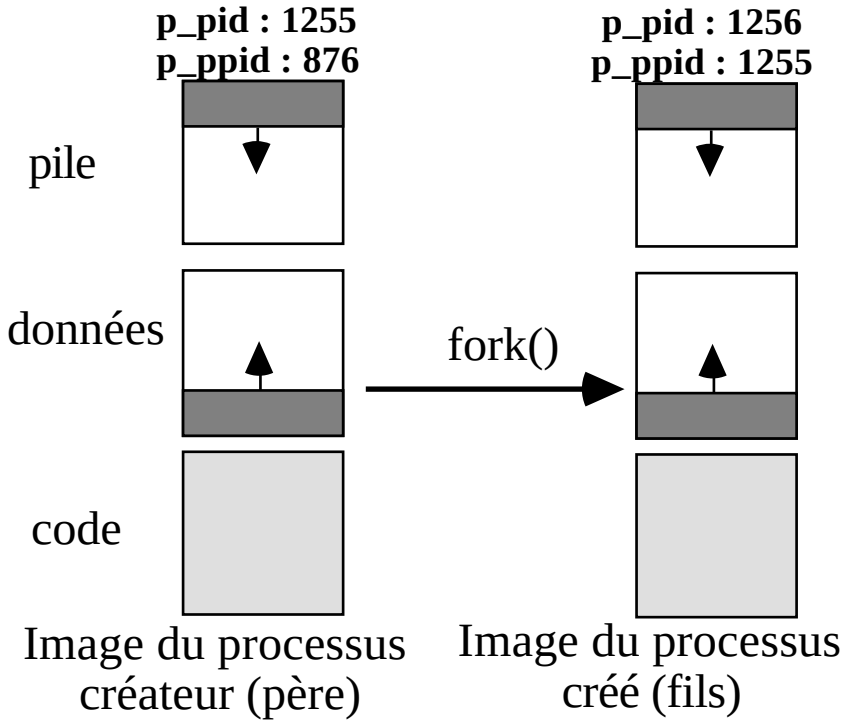
• crée un nouveau processus, qui est un clone du père au moment de la création,

• le fils hérite de tout le contexte du père :

(il le partage donc) même vue du code, même vue des données, mêmes fichiers ouverts, même environnement (variables PATH, TERM,...)

• seuls non hérités : pid et ppid

```
main ()
{
   id = fork();
   if ( id == 0) {                  /*processus fils*/
      printf("je suis le fils : %d\n", getpid());
      exit(0);
   } else {                         /*processus père*/
      printf("je suis le père : %d\n", getpid());
      printf("mon fils a le pid : %d\n", id);
      exit(0);
   }
}
```



En réalité le code n'est pas dupliqué et les données le sont le plus tard possible, technique du copy-on-write ...

Parfois on fait un fork pour changer l'image exécutable immédiatement après, dans ce cas, on utilise la primitive vfork().

Famille de primitives exec

Une primitive de la famille **exec** permet à un processus de charger en mémoire un nouveau code exécutable.

Après un **exec**, le contrôle de l'exécution est donné au programme lancé.

Le programme hérite d'un certain nombre de paramètres du processus qui l'héberge : n° de processus, n° de processus père, l'uid utilisateur réel... Le numéro de processus effectif peut changer si le programme lancé a un bit suid.

Il y a six primitives qui diffèrent

•• par la manière dont les arguments sont passés :

- par liste : execl, execlp, execle
- par tableau : execv, execvp, execve

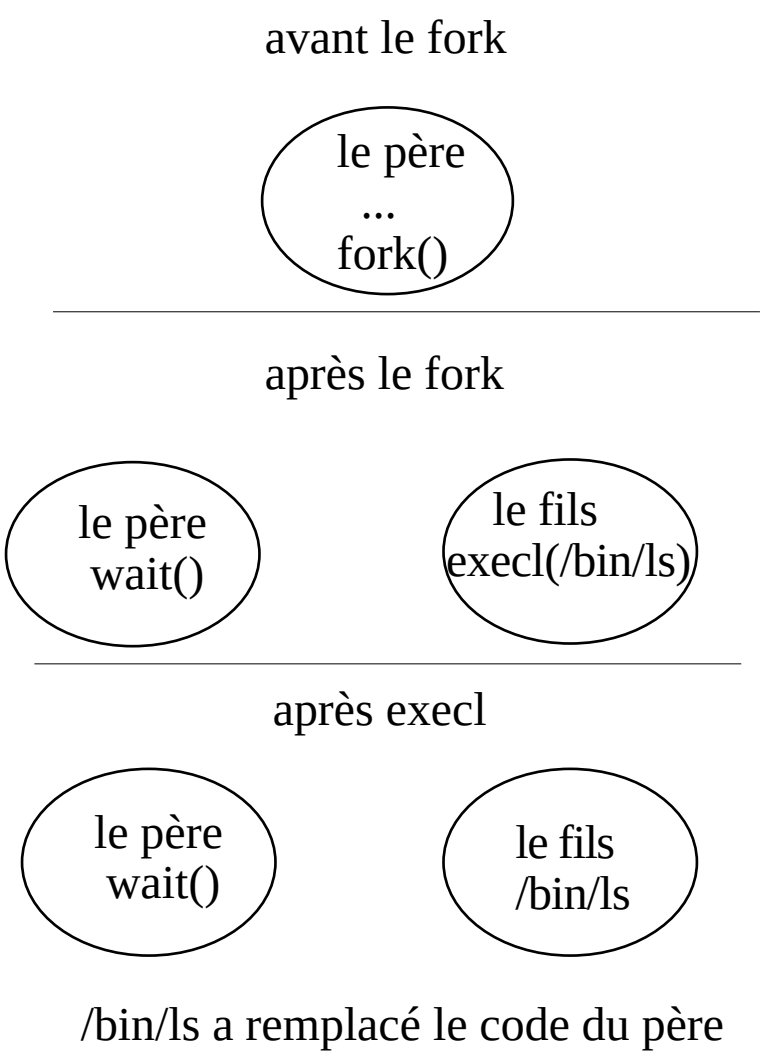
•• par la manière dont est spécifié le chemin d'accès au fichier exécutable à charger

- en utilisant la variable d'environnement **PATH** : execlp, execvp
- relativement au répertoire de travail : execlp, execle, execv, execve

•• par la modification de l'environnement : execve, execle

EXEMPLE DE RECOUVREMENT

```
main ()
{
   id = fork();
   if ( id == 0) {
         /*processus fils, pour la commande ls*/
         execl("bin/ls","ls", "-l", (char*)0);
         perror("erreur de creation par execl()");
   }
   if ( id > 0) {
         wait((int*)0);
            /*le père* attend la fin du fils*/
         printf("execution bien terminee \n");
      exit(0);
   }
   if (id < 0) perror("erreur au fork()");
}
```



Primitive `exit`

Quand il se termine un processus fait `exit(status)`.

Cet appel système a pour effet de passer une information d'état, "status", au système.

Cet état de sortie est récupérable par le processus père à l'aide de l'appel **`wait`**.

• un processus qui se termine passe dans l'état zombie et y reste jusqu'à ce que son père le récupère par un wait(). Le fils disparait alors.

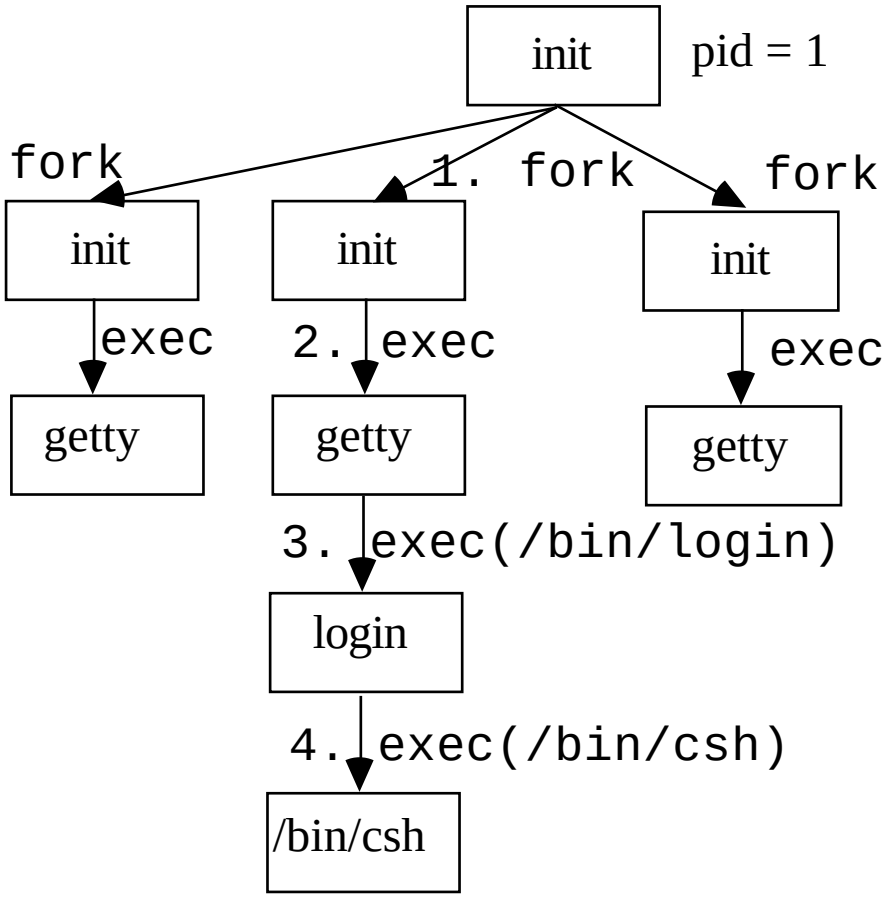
Primitive `wait`

```
int wait (int *status);
```

• Un processus peut récupérer des informations sur un fils qui vient de se terminer grace à **wait**. Il récupère la variable "status" de terminaison, et/ou d'autres données sur le mode de terminaison du processus.

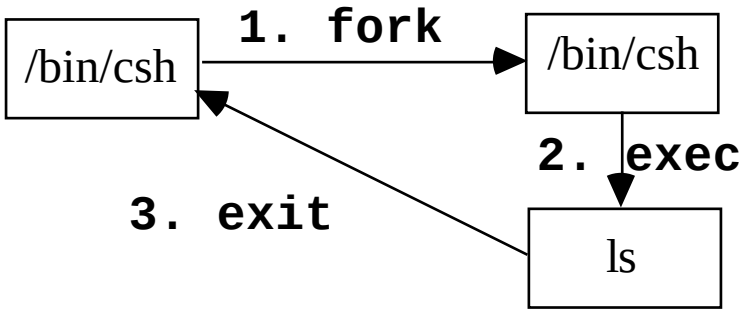
• si un père se termine avant son fils, sans l'attendre, ce fils devient orphelin et est adopté par le processus init, qui ne se termine jamais.

Lancement d'un shell utilisateur

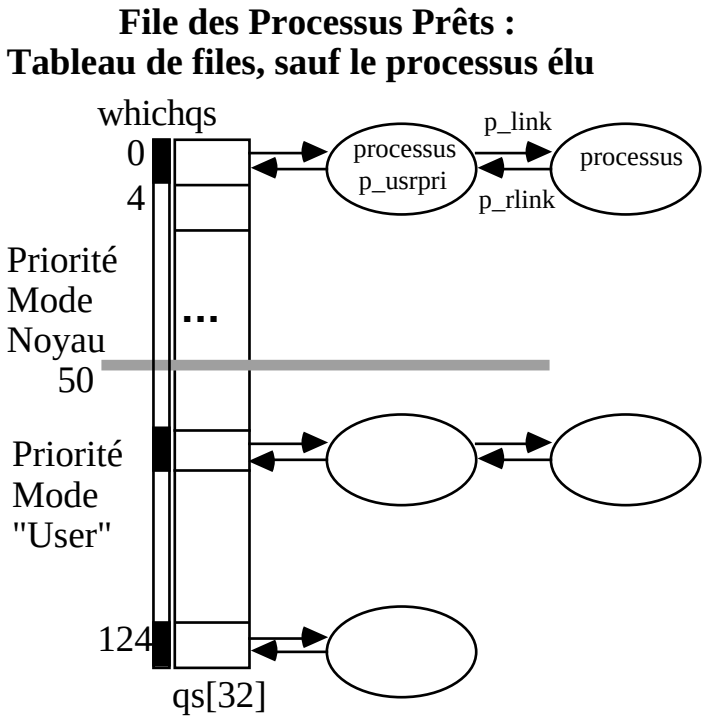


Lancement d'une commande par un interpréteur

Commande ls en terme de processus :



File des processus prêts



La priorité va de 0 à 127 :

0 la plus forte, 127, la plus faible.

Le tableau des files s'appelle `qs[]`. `whichqs` est un vecteur de bits

Ordonnancement dynamique aspects généraux

top horloge noyau : 1 tick (10 ms souvent)

L'ordonnancement s'effectue par priorité dans la file des processus prêts:
• la file de plus forte priorité en premier,
• et dans une file donnée, en tourniquet,
1 quantum de temps correspond à 10 ticks.

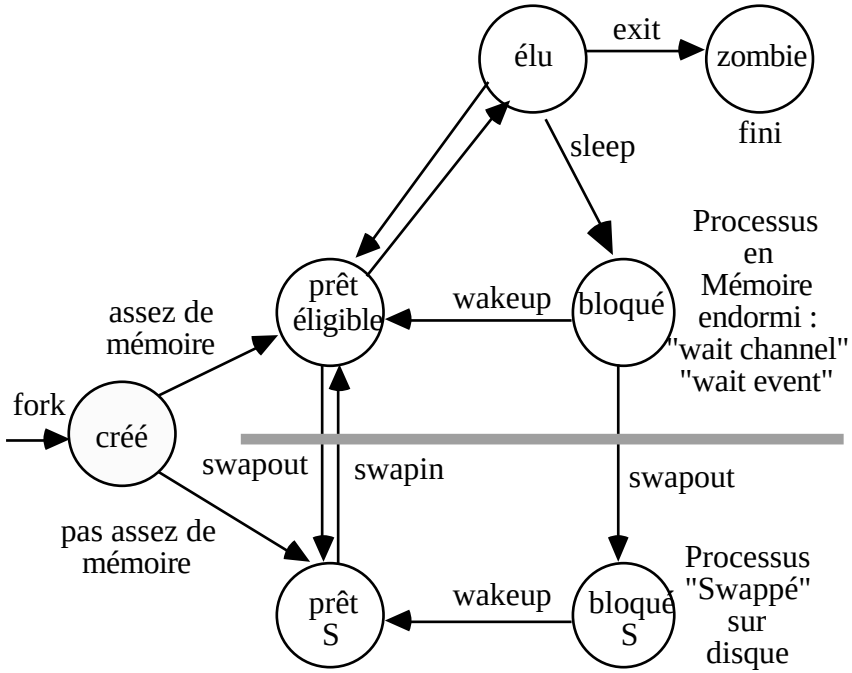
Un processus bloqué n'est pas la file des processus prêts.

Quand un processus épuise son quantum, il est mis en fin de la file dont il vient.

Les processus vont de file en file en fonction de la valeur de leur priorité courante.

Un processus en exécution ne peut être préempté par un processus plus prioritaire qui vient de s'éveiller que s'il effectue un appel système. Sinon, le processus plus prioritaire doit attendre l'épuisement de son quantum.

Etats d'un processus



élu : s'exécute sur un processeur

prêt : s'exécuterait s'il avait le processeur

bloqué : attentes par wait, pour E/S,

Commande ps

délivre la liste des processus et leurs caractéristiques(pid, ppid, état, terminal,..)

```
 USER   PID  STAT  TIME       COMMAND
eric    5015   R    0:00 ps -aux
eric    5000   S    0:00 -csh (csh)
root    3592   S    0:05 rlogind
root     298   I   23:50 /etc/inetd
root       1   S    3:03 init
root    4355   I    0:00 telnetd
root     348   S    0:37 - tty32 (getty)
root     208   I    1:29 /usr/local/cap
root     291   S    0:19 /etc/cron
root       2   D    4:46 pagedaemon
root    9500  IW    0:00 telnet
```

ÉTAT (STAT): séquence de 5 lettres RWNAV:

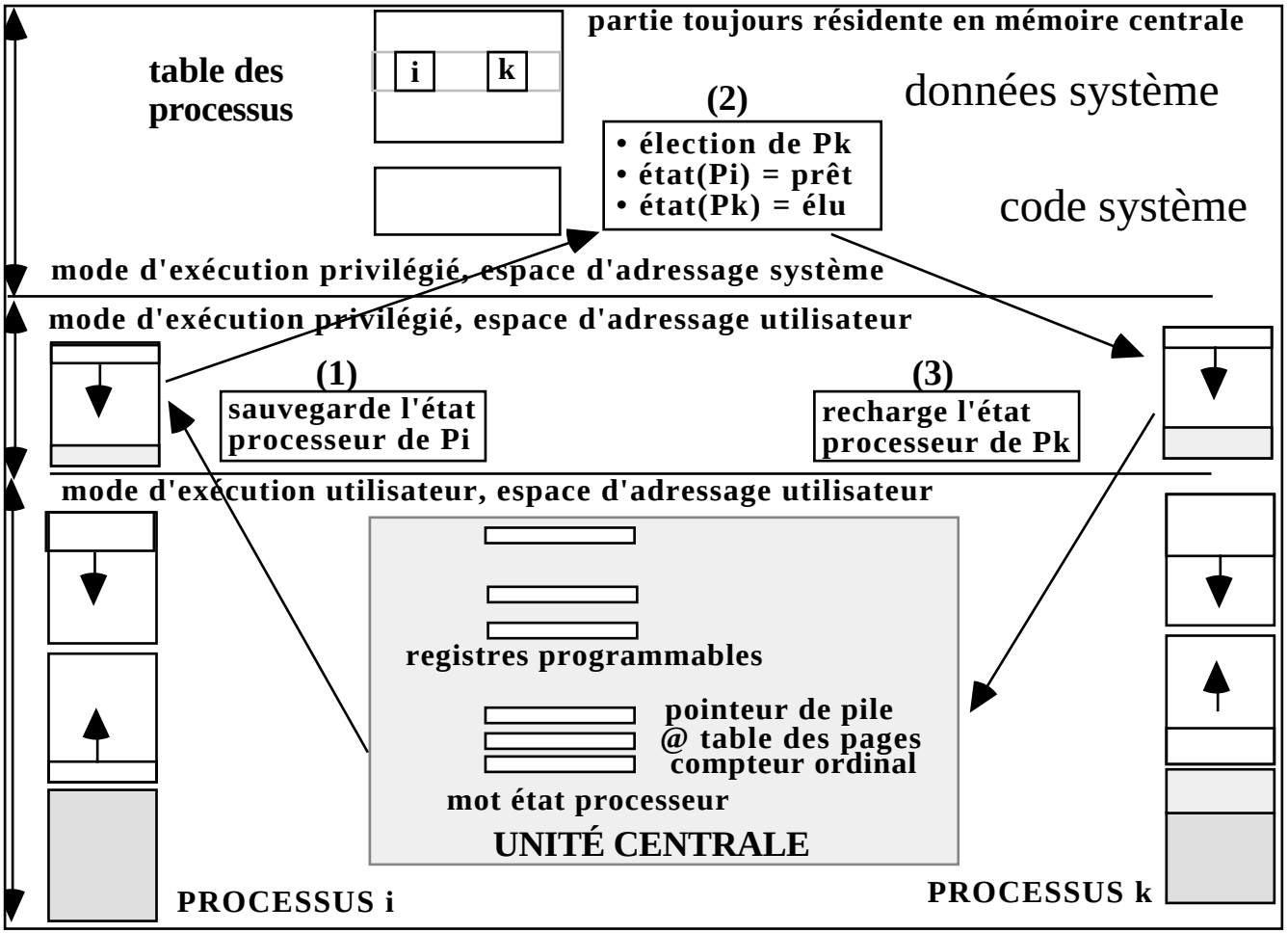
• Première : le processus en cours d'exécution :

R : running, T : stopped, P : en attente de page, D : en attente d'E/S disque, S : sleeping (< 20 secondes), I : idle (> 20 secondes).

• Seconde : processus sur disque (swapped out) :

W : swappé, Z : zombie, <espace> : en mémoire, > : résident en mémoire et a dépassé son quota

• Troisième : priorités ; • Quatrième : mémoire virtuelle ; • Cinquième : réservé



Passage de l'environnement de Pi à celui de Pj

avec sauvegarde et rétablissement du bon contexte processeur