جامعة ابن طفيل Université Ibn Tofail
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⵓⵢⵍ Ibn Tofail University

Système d'Exploitation
-Processus-

Med. AMNAI
Filière SMI-S4
Département d'Informatique

2023-2024

Plan

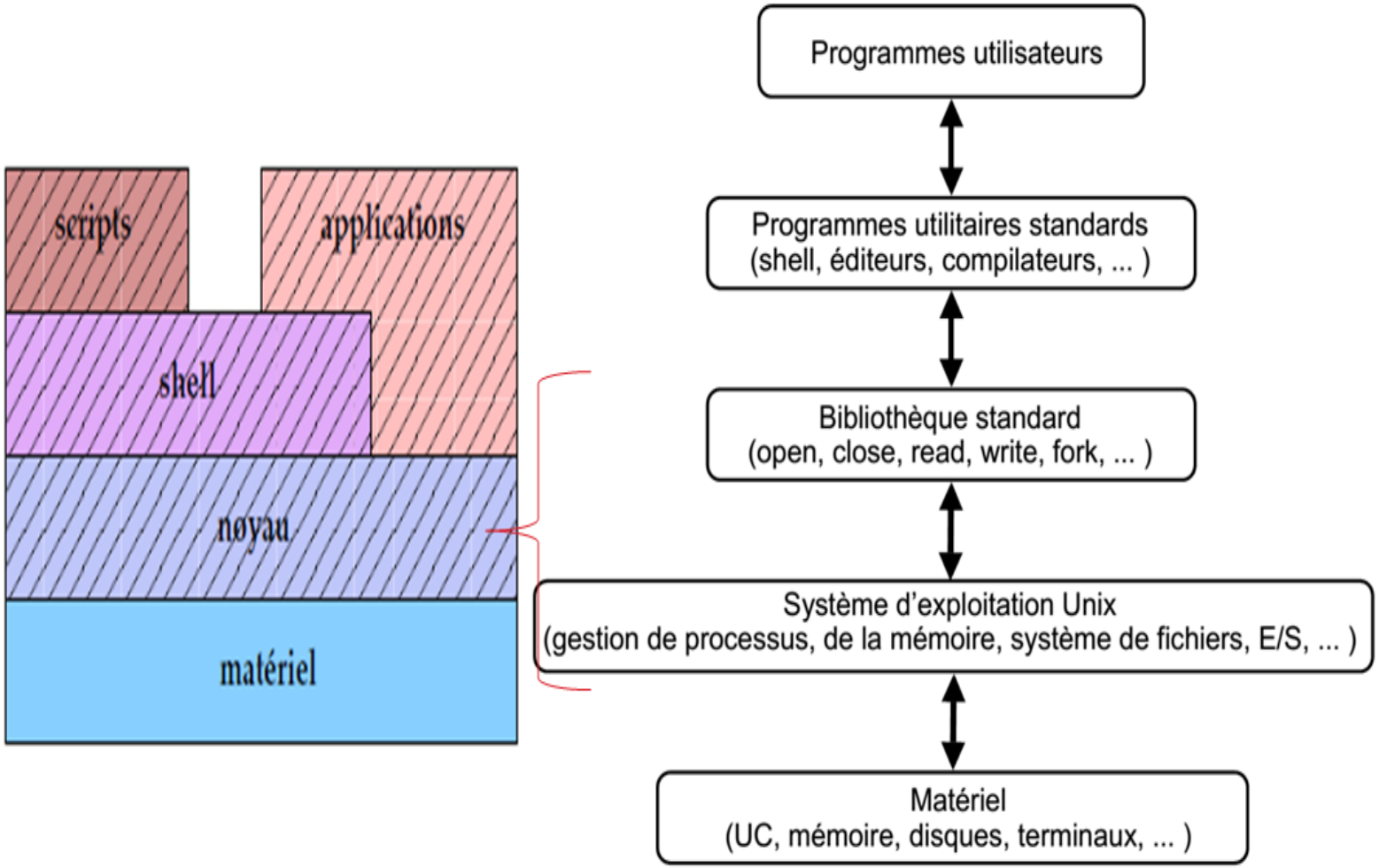
Plan

Plan

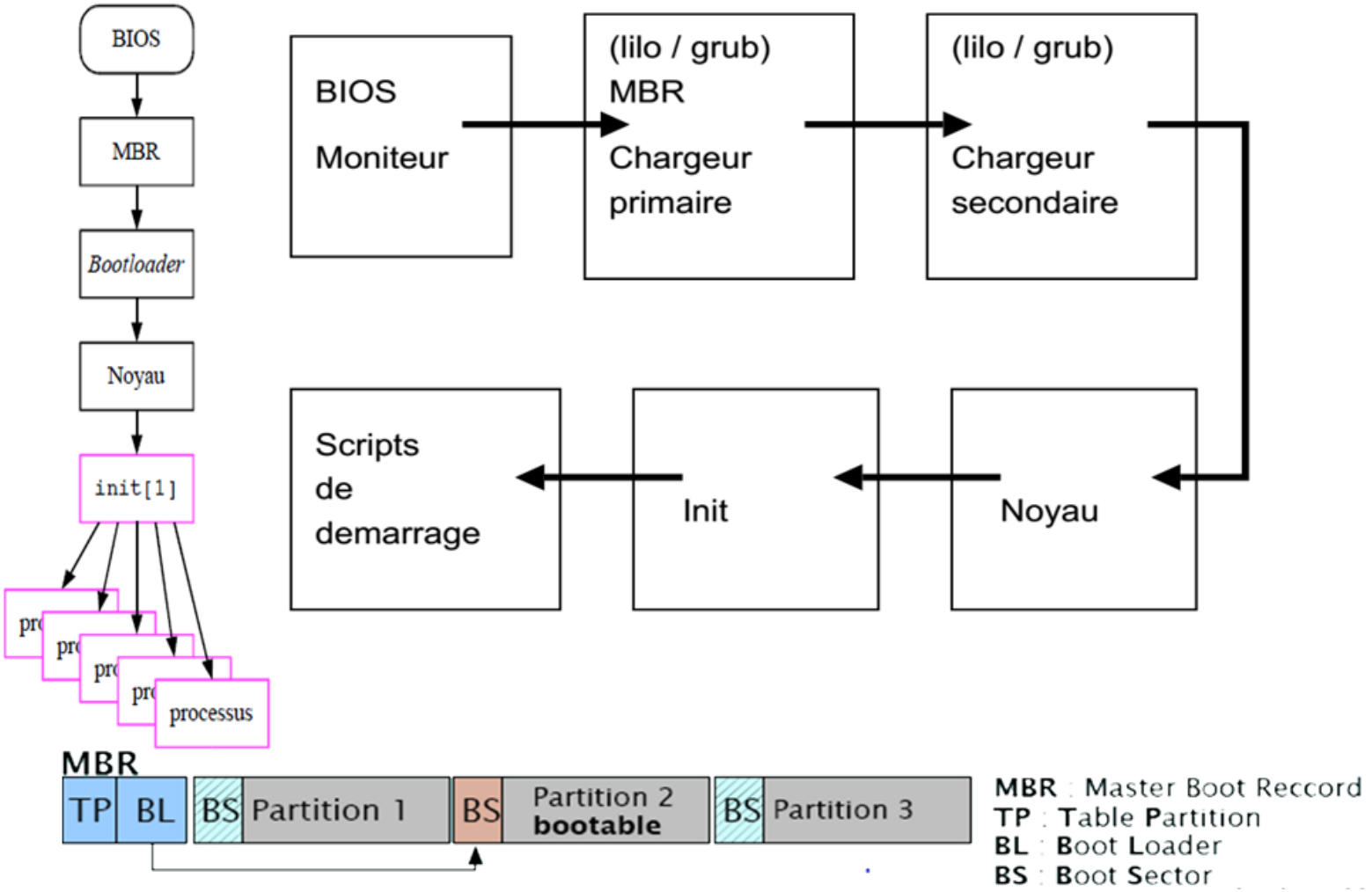
Plan

Plan

Structure Générale d'Unix



Processus de Démarrage



Qu'est-ce qu'un processus ?

- Un processus est un ensemble d'instructions se déroulant séquentiellement sur le processeur ;
- Un processus est l'abstraction d'un programme en cours d'exécution ;
- Chaque exécution d'un programme donne lieu à un processus différent ;
- A tout instant, un processeur exécute au plus un processus.

RQ : Les processus partagent l'accès à différentes ressources : **processeur**, **mémoire** et **périphériques**.

Caractéristiques d'un Processus

Un processu est caractérisé par un **programme**, des **données** et un **contexte** courant d'exécution.

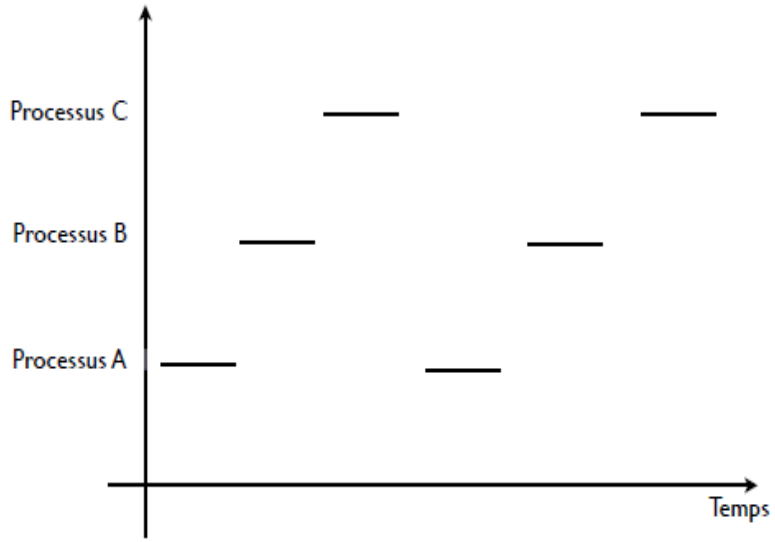
- Son état : exécution, suspendu, etc ;
- Son identificateur ;
- Son compteur ordinal (contrôleur de flux) : indique la prochaine instruction à exécuter ;
- Sa pile d'exécution : en cours de l'exécution, les données obtenues sont stockées dans la pile ;
- Ses données en mémoire ;
- Toutes informations utiles à son exécution (E/S, fichiers ouverts, . . .).

Commande PS

- **PS** : Affiche la liste des processus en cours. Par défaut, elle affiche uniquement ceux de l'utilisateur et du terminal courant.
  - **-u** : Affiche tous les processus de l'utilisateur quelque soit le terminal ;
  - **-A** : Affiche tous les processus du système ;
  - **-f** : Donne plus d'informations sur les processus affichés ;
  - **-H** : Ordonne et indente les processus selon leur généalogie (PPID) ;
  - **TOP** : Affiche les processus consommant le plus de ressource CPU ;
- **ps -ef** : répertorie les processus en cours d'exécution. (Une autre commande similaire est **ps aux**)
- **ps -f –pid id** : Affiche les processus basés sur un **PID**. A la place de *id* entrez le *PID* correspondant.
- **ps aux –sort=-pcpu,+pmem** : Affiche les processus consommant la plus grande quantité de **CPU**.

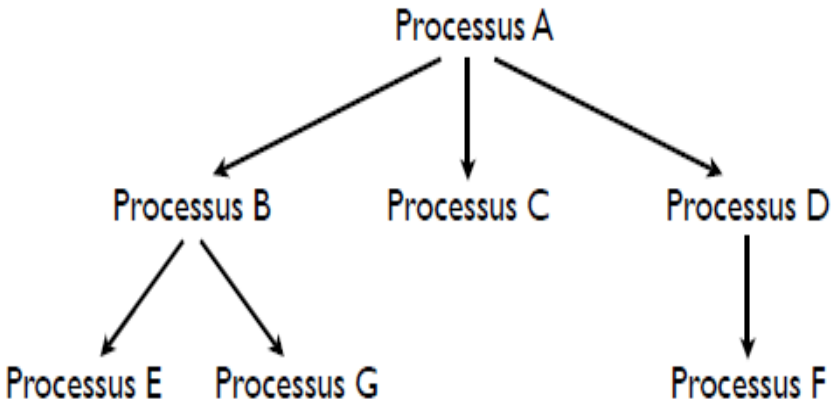
Partage du Temps

- Le processeur alterne entre plusieurs processus en attribuant un **quantum** pour chaqu'un ;
- A tout instant, un seul processus utilise le processeur et un seul programme s'exécute ;
- Le choix du processus suivant est assuré par le OS (**Ordonnancement** ou "Scheduling").



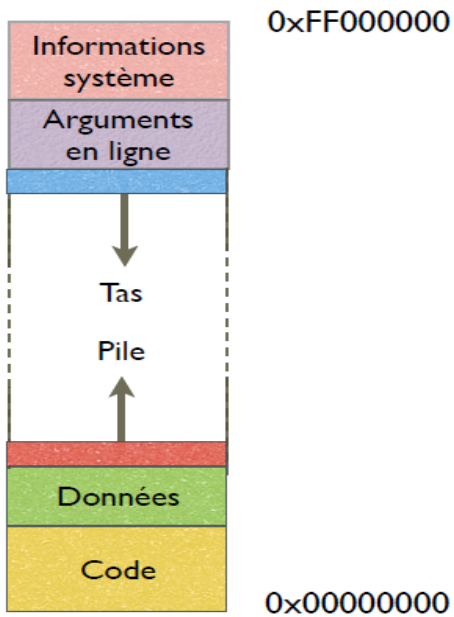
Arborescences de Processus

- Premier processus : **init** ;
- Création d'un processus par duplication d'un autre existant avec appel système **fork()** ;
- Notions de : Processus père et Processus fils.



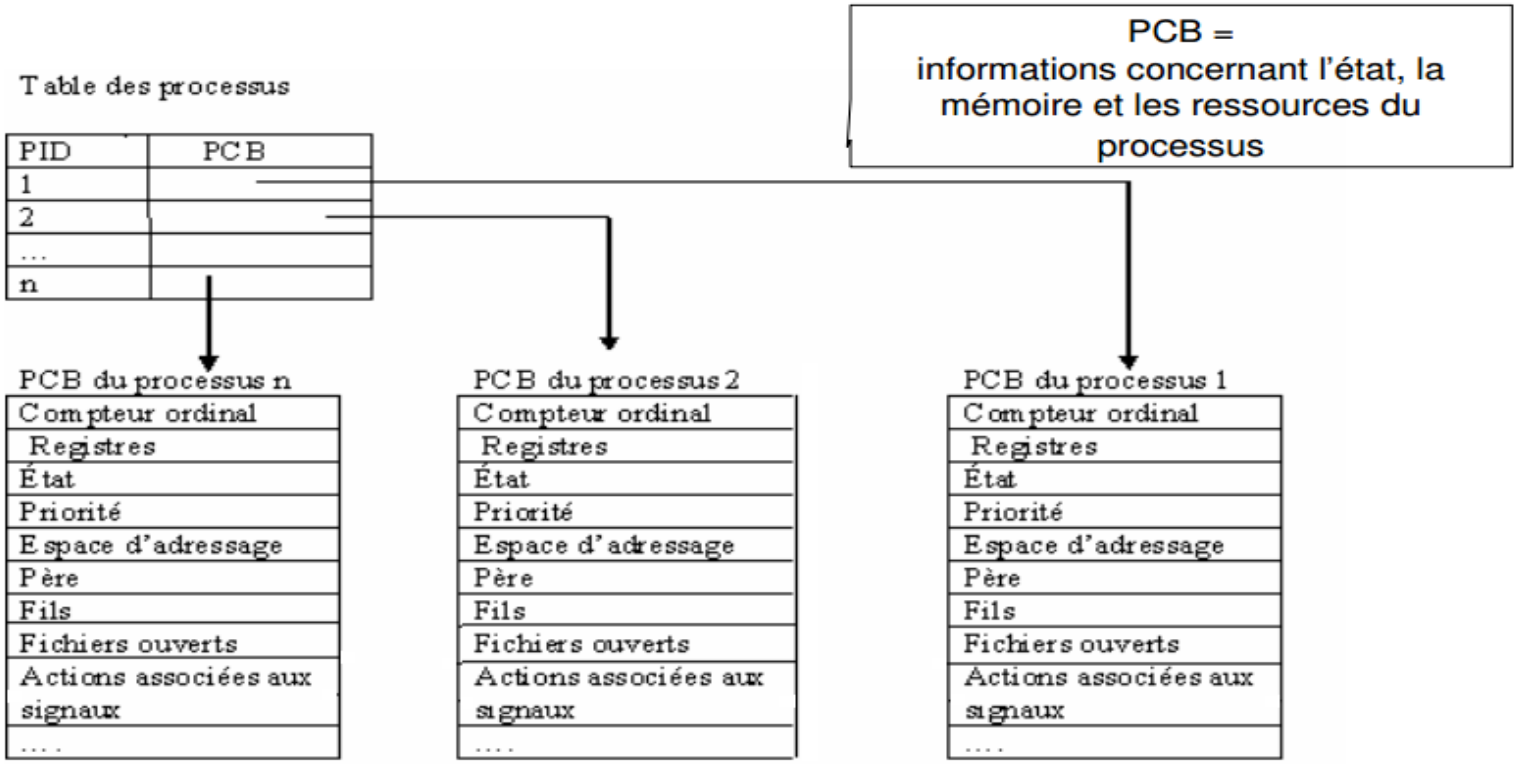
Structures de Gestion des Processus

- Segment de code ;
- Segment de données ;
- Pile (stack) : mémorise, de façon contiguës, les données obtenues en cours de l'exécution ;
- Tas (heap) : stocke les données de façon non contiguë ;
- Bloc de Contrôle de Processus (PCB).



Bloc de Contrôle de Processus (PCB)

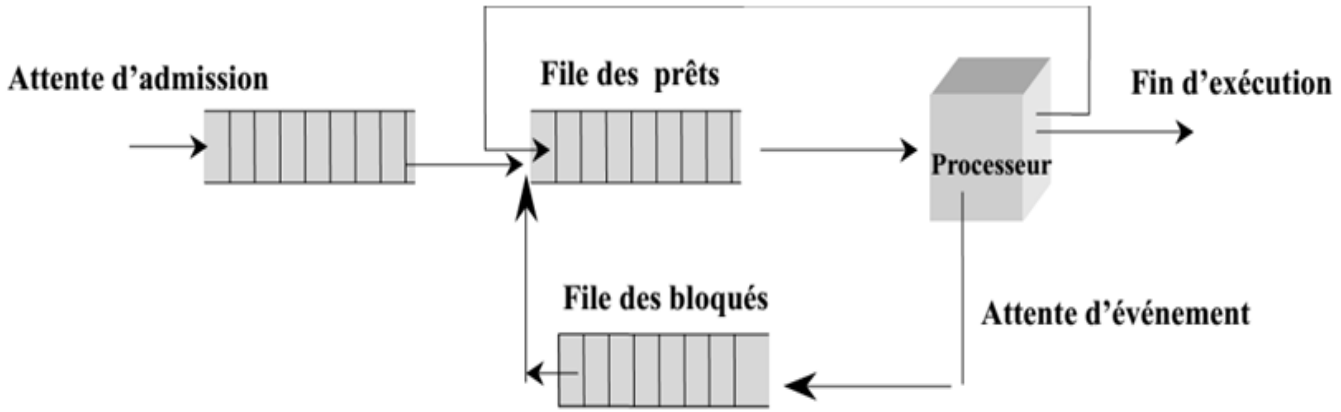
- Le OS maintient toutes les informations sur tous les processus créés dans une table appelée «**table des processus**» ;
- Une entrée par processus : Bloc de Contrôle de Processus (**PCB**).
- Cette table permet au OS de localiser et de gérer tous les processus.

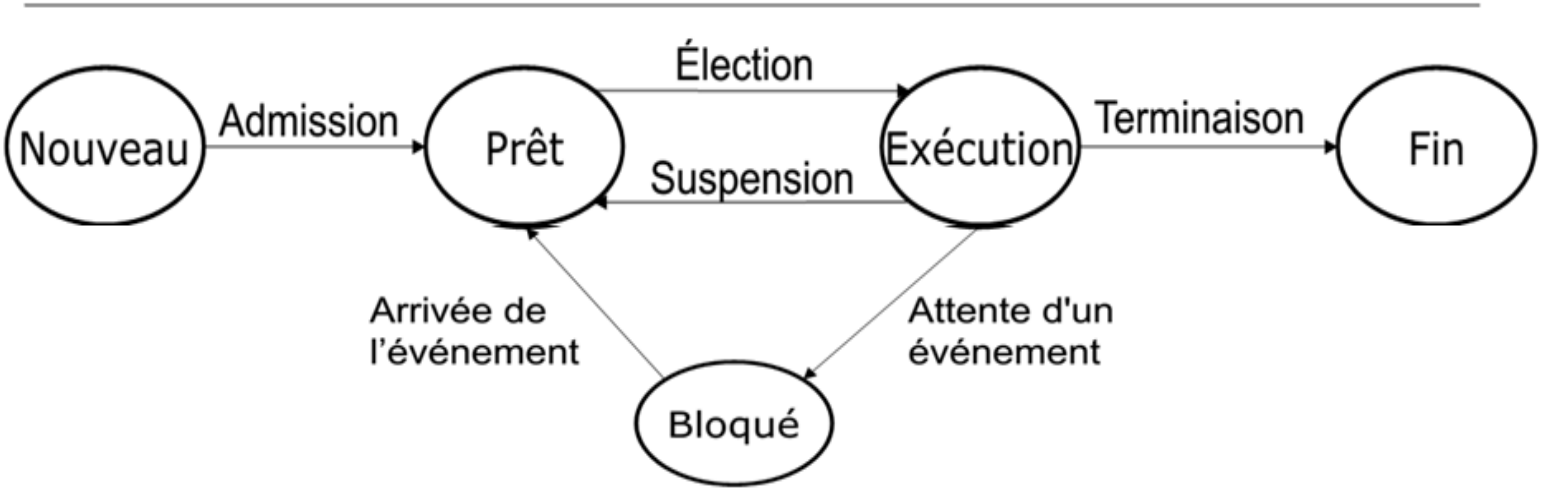


État de Processus

- **Actif ou élu (running)** : le processus utilise le processeur ;
- **Attente ou bloqué** : le processus attend une ressource (ex : fin d'une entrée-sortie).
- **Prêt ou éligible (ready)** : Si le processus n'attend pas une ressource partagée et s'il ne dispose pas du processeur..

Transition des Etats des Processus





Modes d'Exécution

- Mode **noyau** : accès sans restriction ;
- Mode **utilisateur** :
  - accès restreint ;
  - pas d'accès direct aux périphériques ;
  - Il peut être interrompu par d'autres processus.

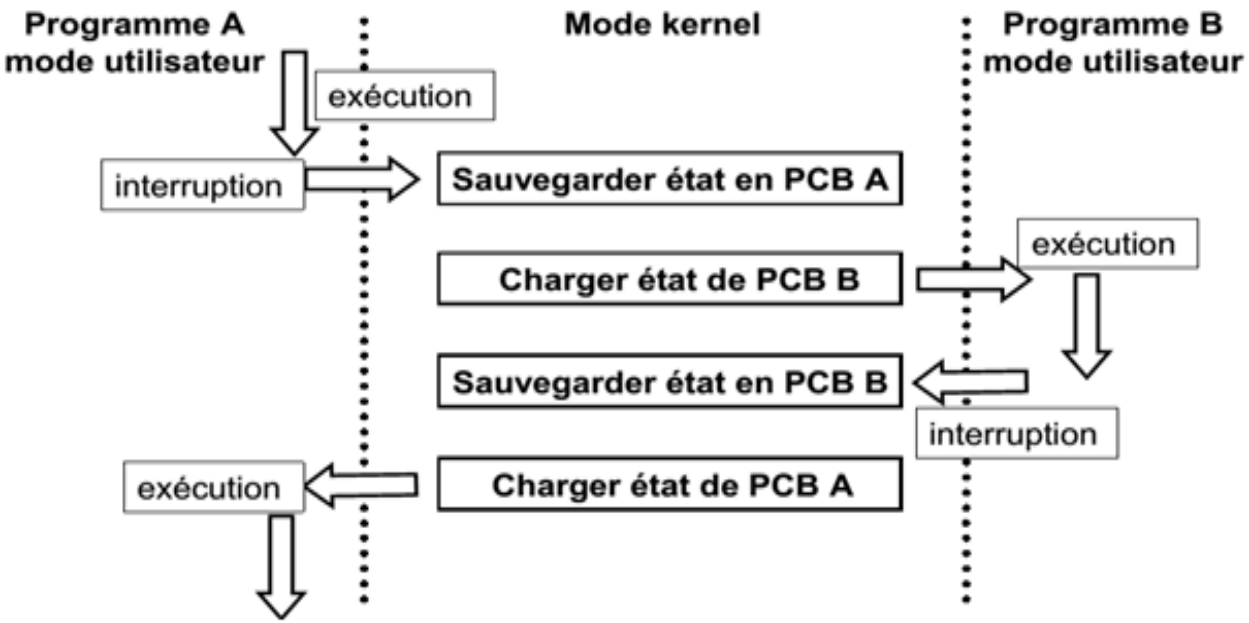
Notion d'Image

Lors de l'exécution d'un programme, ce dernier sollicite un ensemble de composants :

- **Code** ;
- **Données** (Statique, Tas, Pile) ;
- **Contexte** d'exécution ;
  - Pointeur d'instruction.
  - Registres mémoire.
  - Fichiers ouverts.
  - Répertoire courant.
  - Priorité.
  - ...

Changement de Contexte

Le changement de contexte, entre processus, est le passage du mode utilisateur au mode kernel.



Fonctions d'Identification des Processus

Plusieurs identificateurs sont associés à un processus :

- Numéro de processus (**pid**) : **getpid(void)** ;
- Numéro du processus père (**ppid**) : **getppid(void)** ;
- Identificateur d'utilisateur réel ;
- Identificateur d'utilisateur effectif (bit **setuid**) ;
- Identificateur de groupe réel ;
- Identificateur de groupe effectif (bit **setgid**) ;
- Liste d'identificateurs de groupes.

Exemple

Pid du **processus** et de son **père**.

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

int main(int argc, char *argv[])
{
    printf("mon pid est : %d\n", (int)getpid());
    printf("le pid de mon pere est : %d\n", (int)getppid());
    exit(EXIT_SUCCESS);
}
```

Création de Processus

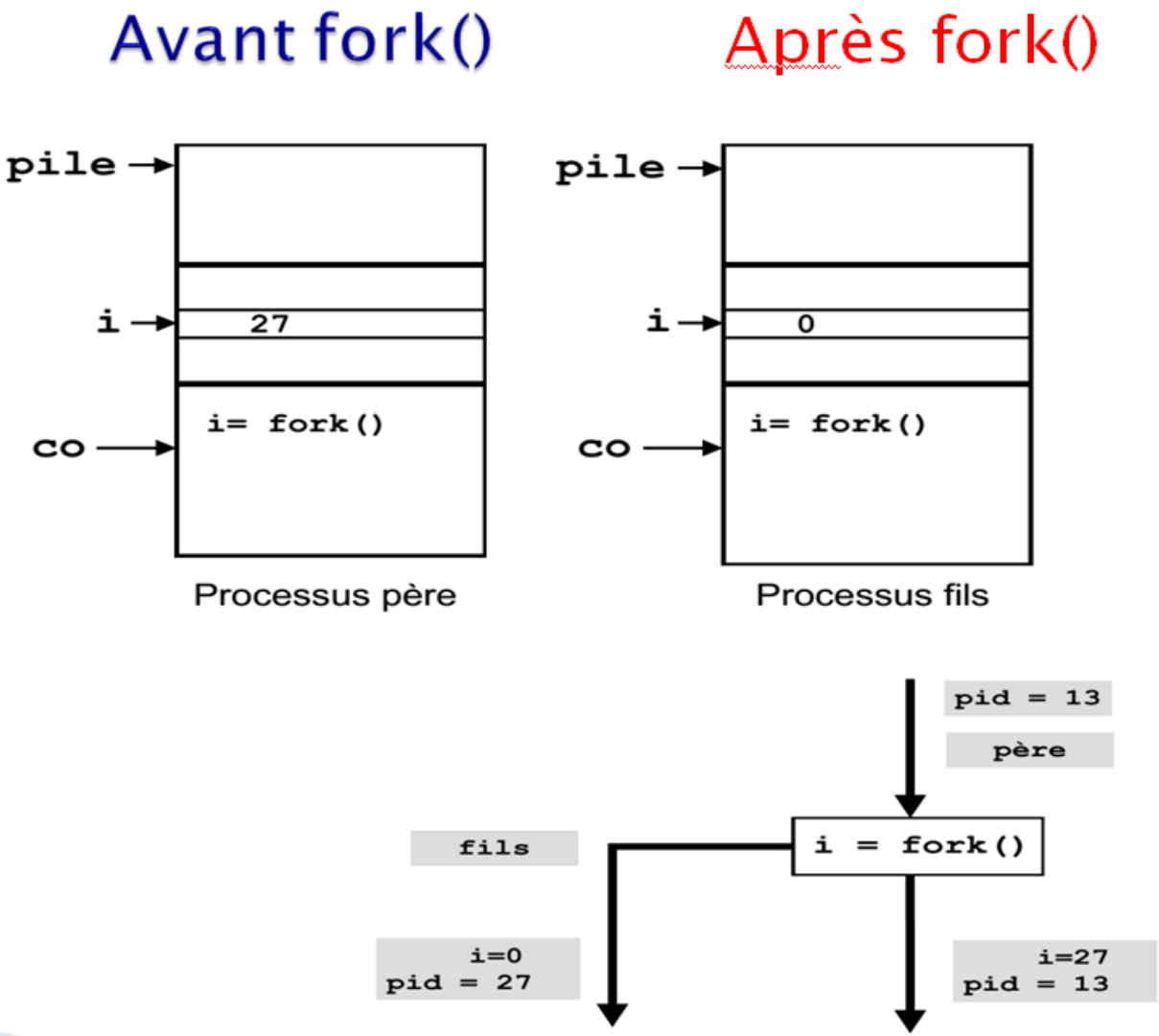
- **pid_t fork(void)** : La valeur retournée est le **pid** du fils pour le processus père, ou **0** pour le processus fils. La valeur **-1** est retournée en cas d'erreur.
- Recopie totale (données, attributs) du processus père vers son processus fils (nouveau pid).
- Le fils continu son exécution à partir de cette primitive.
- Héritage du fils :
  - La priorité ;
  - La valeur du masque ;
  - Le propriétaire ;
  - Une copie des descripteurs des fichiers ouverts ;
  - Le pointeur de fichier (offset) pour chaque fichier ouvert ;
  - Le comportement vis à vis des signaux.

Exemple fork()

```c
#include<stdio.h>//ex1
#include <sys/types.h>
#include <unistd.h>

int main()
{
        int f;
        f = fork();

        printf ("Valeur retournee par la fonction fork : %d\n",(int)f );
        printf ("Je suis le processus numero %d\n",(int)getpid() );

return 0;
}
```

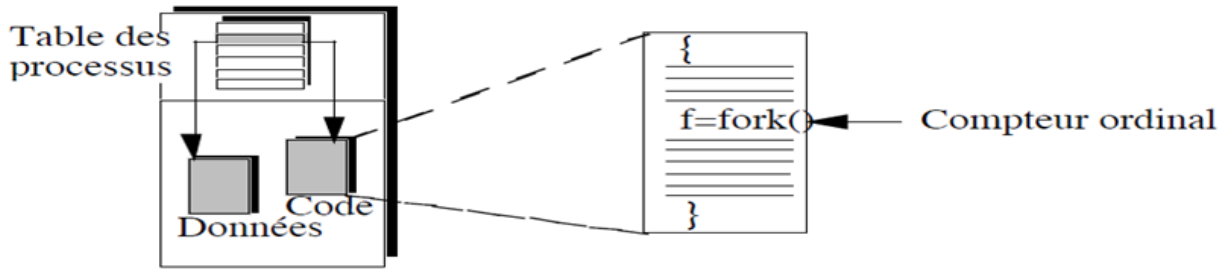
```
[root@localhost app]# ./ex1
 Valeur retournee par la fonction fork: 3952
 Je suis le processus numero 3951
 Valeur retournee par la fonction fork: 0
 Je suis le processus numero 3952
```

Exemple fork() (suite)

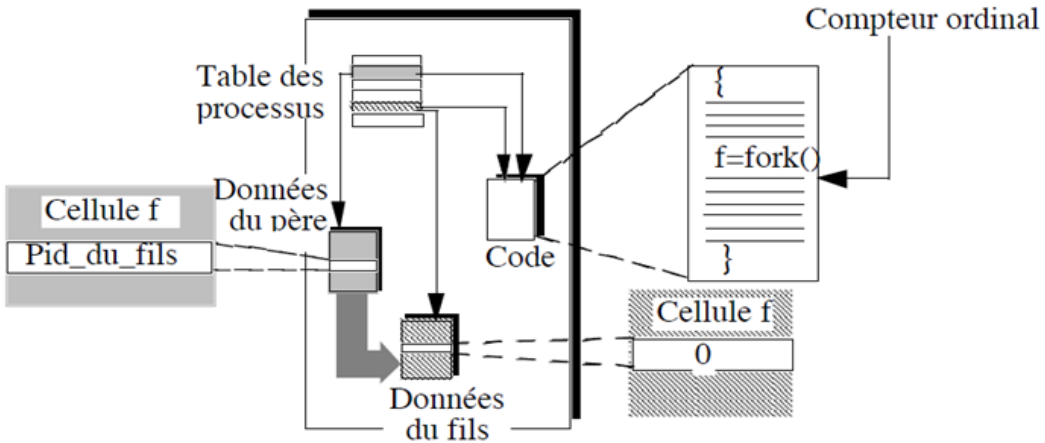


Exemple fork() (suite)

Avant fork()



Après fork()



Exemple : père et fils

```
int main()//ex2
{
        int f;
        f = fork();
        if(f == -1){
                printf ("Erreur : le processus ne pas etre cree");
        }
        if(f == 0){
                printf ("Ici le processus fils \n");
                printf ("Je suis le processus numero %d\n",(int)getpid() );
        }
        if(f != 0){
                printf ("Ici le processus pere \n");
                printf ("Je suis le processus numero %d\n",(int)getpid() );
        }
return 0;
}
```

```
[root@localhost SharedVirtuaMachines]# ./ex2
Ici le processus pere
Je suis le processus numero 28468
[root@localhost SharedVirtuaMachines]# Ici le processus fils
Je suis le processus numero 28469
```

Exemple : Portée du code

```
[root@localhost SharedVirtuaMachines]# ./ex3
Je suis seul au monde
Je suis le processus pere
Et la qui suis-je  28592
[root@localhost SharedVirtuaMachines]# Ici le processus fils
Et la qui suis-je  28593
```

```
int main()//ex3
{
        int f;
        printf ("Je suis seul au monde \n");
        f = fork();
        if(f == -1){
                printf ("Erreur fork()\n");
        }
        if(f == 0){
                printf ("Ici le processus fils \n");
        }else{
                printf ("Je suis le processus pere \n");
        }
        printf (« Et la qui suis-je  %d\n", (int)getpid() );
return 0;
}
```

Exemple : Portée des Variables

```
int main(){//ex4
        int i,j,f;
        i=5;j=2;

        f = fork();
        if(f == -1){
                printf ("Erreur fork()\n");
        }
        if(f == 0){
                //code du fils
                printf ("fils %d\n",getpid());
                j--;
        }
        else{
                printf ("pere \n",getpid());
                i++;
        }
        printf ("Pid : %d i:%d j:%d\n", getpid(),i,j);
}
```

```
[root@localhost SharedVirtuaMachines]# ./ex4
pere
Pid : 28644 i:6 j:2
[root@localhost SharedVirtuaMachines]# fils 28645
Pid : 28645 i:5 j:1
```

Synchronisation

- Un processus père est toujours prévenu de la fin d'un fils et le fils est toujours prévenu de la fin du père.
- Si la fin d'un fils n'est pas traitée par le père, il devient (fils) processus **zombie**.
- **Synchronisation** pour les fins d'exécution :
  - Possibilités d'échanges d'informations permettant une synchronisation sur les fins d'exécutions.
  - Eviter les processus zombies (fin propre).
  - Nécessite que le père soit en attente (**wait()**).
  - Information de la fin du processus fils (**exit()**).

Appel système exit()

void exit(int status) ; termine normalement un processus et la valeur **status** est envoyée au processus père ;

- Un **exit(0)** signifie que le programme s'est exécuté sans erreur :
- **Exit (int)**
  - **EXIT_SUCCESS** constante qui vaut 0
  - **EXIT_FAILURE** constante qui vaut 1
  - Valeur du **int** est "transmise" au père : code de retour.
  - Différentes valeurs peuvent désigner différents types d'erreurs.

Appel système Wait()

wait() permet de suspendre l'exécution du père jusqu'à ce que l'un de ses enfants se termine.

- **pid_t wait(int *satatus)**
- **En cas de réussite** : la valeur retournée est le **pid** du fils qui s'est **terminé**.
- **En cas d'erreur** : la valeur retournée est **-1**.
- L'entier pointé enregistre l'état du fils lorsqu'il est finit (**valeur en paramètre dans exit()**)

Exemple : Appels système wait et exit

```
#include<stdlib.h>//ex6
#include<sys/wait.h>
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){ int r,s,w;

        if((r = fork()) == 0){
                printf("Fils %d\n",getpid());
                //Traitement long
                exit(14);
        }else{
                //Père doit attendre la mort de son fils
                w=wait(&s);
                printf("w : %d s:%d \n",w,s);
        }

return 0;
}
```

```
[amnai@localhost Documents]$ ./ex6
Fils 8938
w : 8938 s:3584
```

Mise en sommeil d'un processus : sleep()

Int sleep(int n) : La fonction suspend l'exécution du processus appelant pour une durée de **n** secondes.

```
int main()//ex7
{
        int PID, status;
        printf("processus pere debut %d\n",getpid());

        if(fork() == 0){
                printf("processus fils %d\n",getpid());
                exit(10);
        }
        PID=wait(&status);
        printf("processus pere %d\n", getpid());
        printf("sortie du wait \n");
        sleep(15);
        printf("PID = %d status = %d \n", PID, status);
        exit(0);
}
```

```
[root@localhost SharedVirtuaMachines]# ./ex7
processus pere debut 29473
processus fils 29474
processus pere 29473
sortie du wait
PID = 29474 status = 2560
```

Attente de 15s avant l'affichage de la derniére line

Recouvrement d'un processus

Un processus peut changer de code par un appel système à **exec** :

- Code et données remplacés ;
- Pointeur d'instruction réinitialisé.

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ...,const char *argn, char * /*NULL*/);
```

path = nom de l'exécutable recherche dans $PATH
arg0 = nom exécutable affiché par PS
argn = n − 2ieme argument de l'executable
NULL = argument de fin de la ligne de commande

Exemple : Appels système execl()

```
int main()//ex8
{
        printf("Je suis un programme qui va exécuter /bin/ls -l \n");

        execl("/bin/ls","ls","-l",NULL);
}
```

```
[root@localhost SharedVirtuaMachines]# ./ex8
Je suis un programme qui va executer /bin/ls -l
total 116
-rwxrwxrwx. 1 amnai amnai  265 11 janv. 11:38 1_fork.txt
-rwxrwxrwx. 1 amnai amnai  438 10 janv. 16:37 2_Exemple_fork.c
```

Appels système execl() (bis)

Lorsqu'il s'exécute **execl**, il est totalement remplacé par l'exécution du "**ls -l**", le processus avec le programme, disparaît.

```
int main()//ex9
{
        printf("Je suis un programme qui va executer /bin/ls -l \n");

        execl("/bin/ls","ls","-l",NULL);

        printf("Je suis le programme qui a execute /bin/ls -l \n");

        //Code qui ne sera pas exécuté !!!!!
}
```

```
Je suis un programme qui va executer /bin/ls -l
total 140
-rwxr-xr-x. 1 root    root    6504 Mar  7 05:51 a.out
-rwxrwxr-x. 1 othmane othmane 6668 Mar  7 17:48 exec1
-rw-rw-r--. 1 othmane othmane  147 Mar  7 17:47 exec1.c
-rw-rw-r--. 1 othmane othmane  149 Mar  7 17:47 exec1.c~
-rwxr-xr-x. 1 root    root    6937 Mar  7 07:09 Exemple1
-rw-rw-r--. 1 othmane othmane  170 Mar  7 07:09 Exemple1.c
-rw-rw-r--. 1 othmane othmane  171 Mar  7 07:08 Exemple1.c~
...
```