Chapitre II : Gestion des Processus

Cours Système Exploitation II

1 LF info ISIMM

Dr. Sana BENZARTI

2024/2025

Contenu du cours

1– Qu'est ce qu'un processus ?

2– Cycle de vie d'un processus

3– Le PCB

4– Contexte d'exécution d'un processus

5– l'appel système fork()

Définition d'un Processus

Un processus est un programme en cours d'exécution ; exemple :

- **Un programme d'application**
- **Un script**
- **Tout autre type de code exécutable**

Un processus — **Créé par** → **Système d'exploitation**

Un autre processus existant

Définition d’un Processus

Chaque processus possède un identifiant unique appelé PID (Process ID), qui permet de l’identifier et de le manipuler

Le système d’exploitation est responsable de la gestion des processus en les exécutant, en les arrêtant, en les mettant en attente ou en les planifiant pour l'exécution.

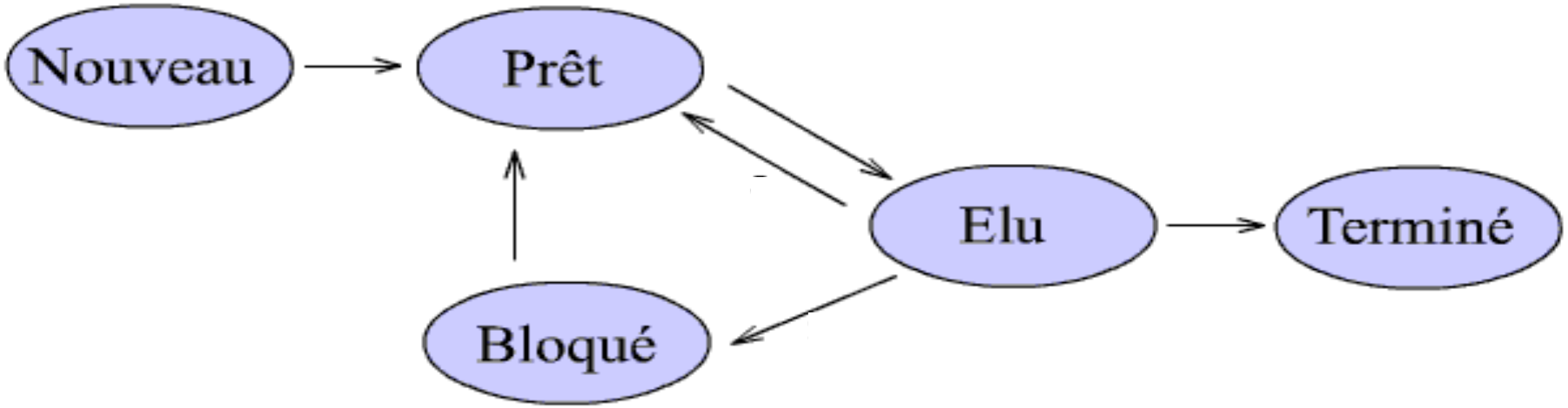
Différence entre programme et processus

- **Un programme est une entité passive : un code écris en langage C qui comporte une liste d'instructions.**

- **Un processus est une entité active avec un ensemble de ressources associées.**

- **Donc un programme devient un processus lorsqu'un fichier exécutable est charge en mémoire.**

Cycle de vie d'un processus



1. **Nouveau : le processus vient juste d'être créé et attend d'être affecté à un processeur.**

2. **Prêt : le processus est prêt à s'exécuter et attend d'être affecté à un processeur disponible.**

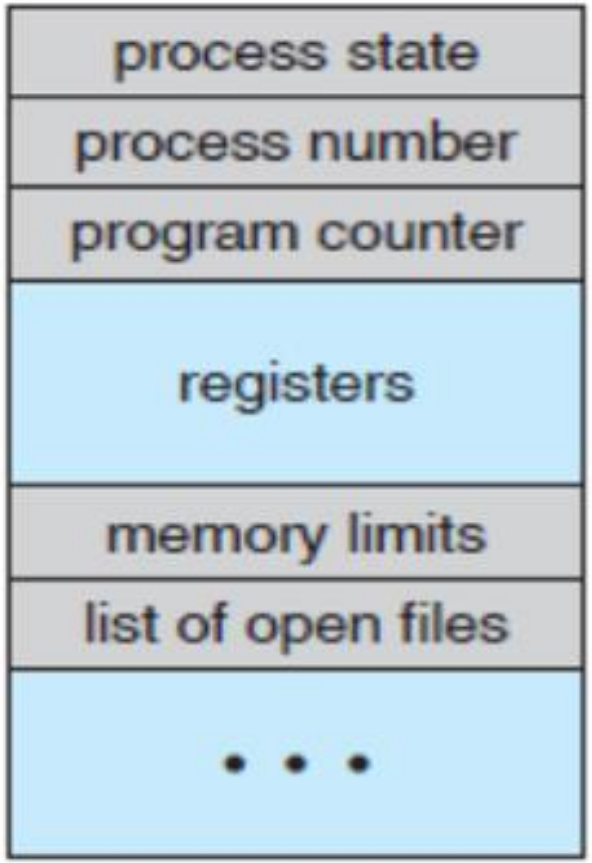
3. **En cours d'exécution : le processus est en train d'être exécuté sur un processeur.**

4. **En attente (ou bloqué) : le processus est en attente d'un événement ou d'une ressource, comme une entrée/sortie ou une réponse à une demande de service.**

5. **Terminé : le processus a terminé son exécution et attend d'être supprimé de la liste des processus actifs.**

Process Control Block PCB

Chaque processus est représenté dans le système d'exploitation par un bloc de contrôle de processus (Process Control Block PCB) appelé aussi un task control block

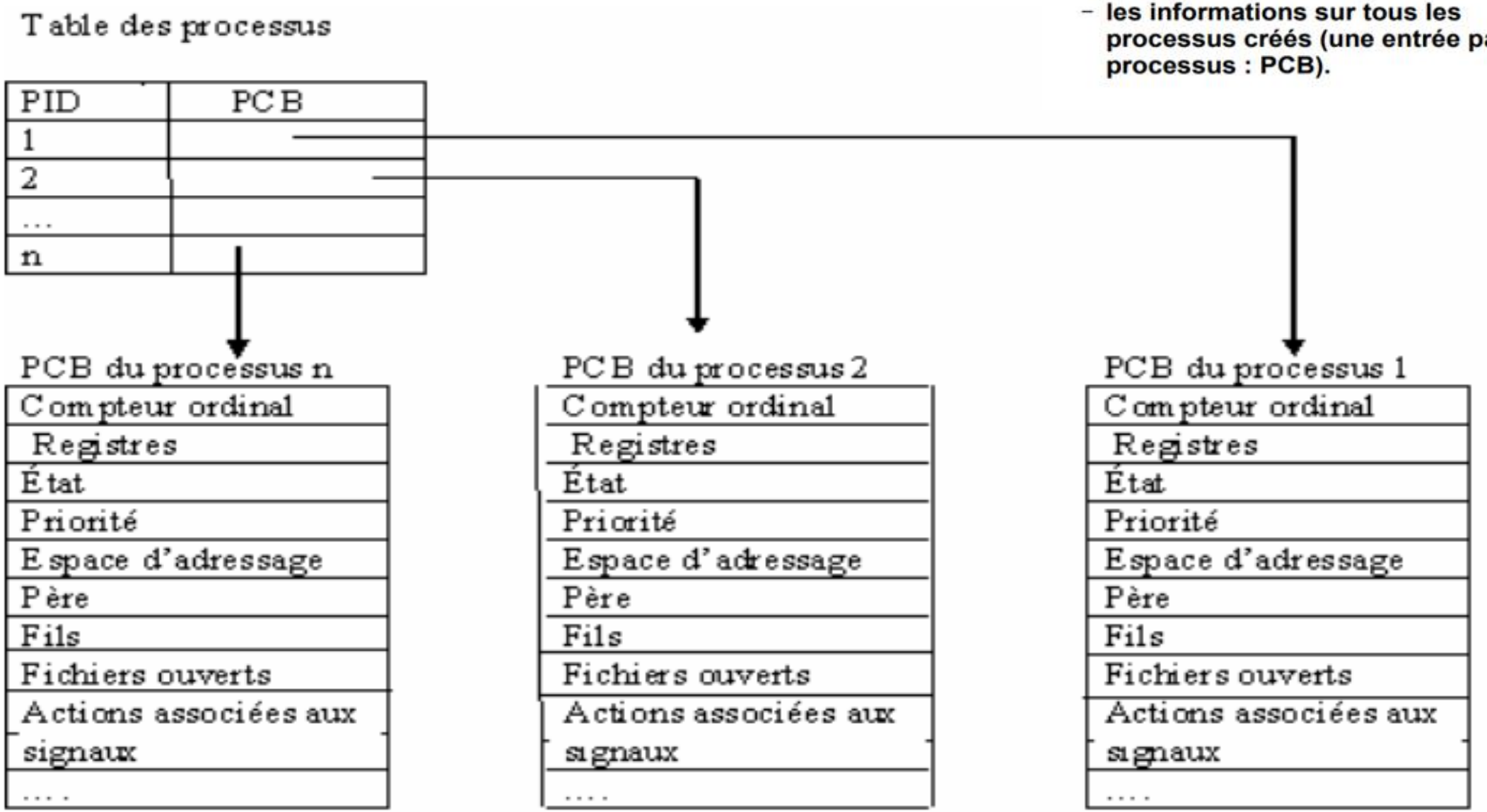


Process Control Block PCB

- **Le bloc de contrôle d'un processus (PCB) contient les informations suivantes :**
  - **un identificateur unique** du processus (un ‘entier’) : le PID (process identifier)
  - **l'état courant du processus** (élu, prêt, bloqué)
  - **le contexte processeur du processus** : la valeur du CO, la valeur des autres registres du processeur
  - **le contexte mémoire** : ce sont des informations mémoire qui permettent de trouver le code et les données du processus en mémoire centrale
  - **des informations diverses** de comptabilisation pour les **statistiques** sur les performances système
  - des **informations liées à l' ordonnancement du processus**. Le PCB permet la sauvegarde et la restauration du contexte mémoire et du contexte processeur lors des opérations de commutations de contexte .

Process Control Block PCB

- **Le système d'exploitation maintient dans une table appelée «table des processus»**
  - **les informations sur tous les processus créés (une entrée par processus : PCB).**



Processus sous Linux

La qualité du fonctionnement **multitâche** d'Unix représente l'un des points les plus attractifs de ce système d'exploitation.

- On peut faire exécuter simultanément plusieurs programmes sans qu'aucun d'eux ne ressente la présence des autres, ou à l'inverse en leur permettant de dialoguer entre eux.

Processus sous Linux

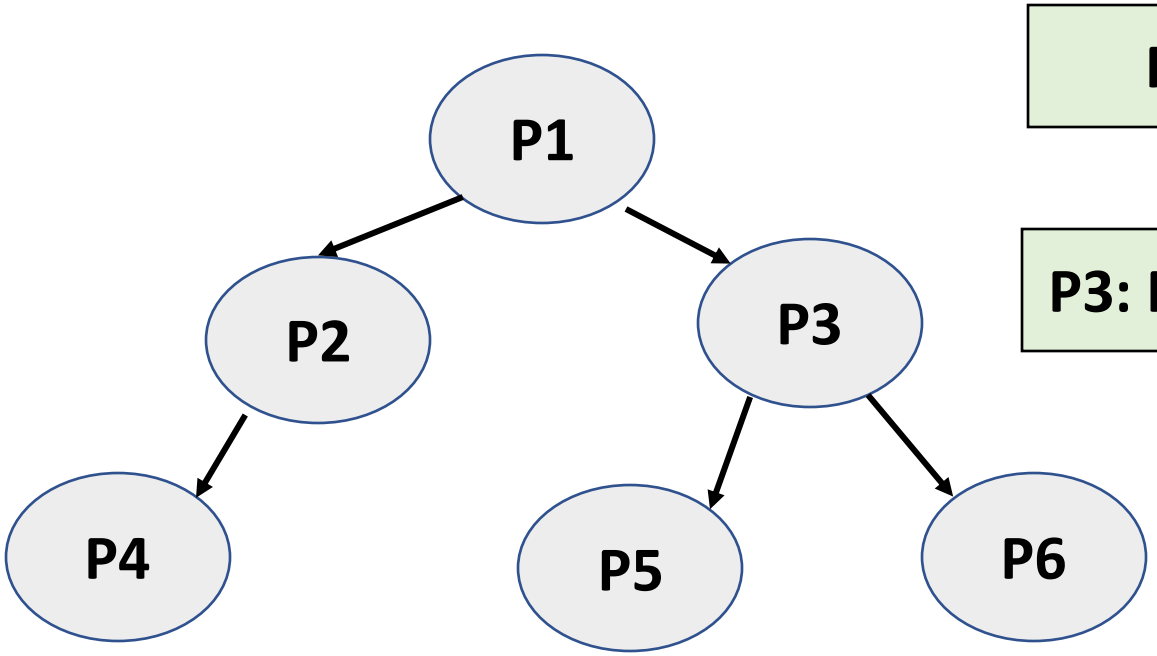
Le système d'exploitation fournit un ensemble d'appels système qui permettent la création, la destruction, la communication et la synchronisation des processus dynamiquement.

Le processus créateur est appelé processus père

Le processus créé est appelé processus fils

Processus sous Linux

Un processus peut créer un ou plusieurs processus fils qui, à leur tour, peuvent créer des processus fils sous une forme de structure arborescente



P1 : Ancêtre

P3: Père de P5 et P6 et fils de P1

P5 et P6 : Fils de P3

Appel système fork()

- Le parallélisme Unix bas niveau est fourni par le noyau qui **duplique** un processus lorsqu'on invoque **l'appel–système fork**() .

- Les deux processus sont alors strictement **identiques**. Le processus fils est la copie exacte du processus père, ils partagent le même code, le segment de données variables).

Les deux processus s'exécutent d'une manière **concurrente** (**en parallèle**).

Appel système fork()

- Un nouveau processus est créé par fork() du système appelant.

- Le nouveau processus comprend une copie de l'espace d'adressage du processus original.

- Les deux processus (le parent et l'enfant) poursuivent l'exécution à l'instruction après le fork(), avec une différence:

– le code retour du fork() est égal à zéro chez le nouveau (enfant) processus,
– alors que le PID (non nulle) de l'enfant est retourné au processus père.

Appel système fork()

- Deux valeurs de retour en cas de succès:
  - Dans le processus père : valeur de retour = le PID du fils,
  - Dans le processus fils : valeur de retour = zéro.

- Sinon
  - erreur : valeur de retour = -1.

- Afin d'obtenir le numéro du processus, il suffit de faire l'appel système **getpid**(), ou **getppid**() pour obtenir le numéro du père.
  - PID (Process IDentifier)
  - PPID : numéro du processus père (Parent Process Identifier)

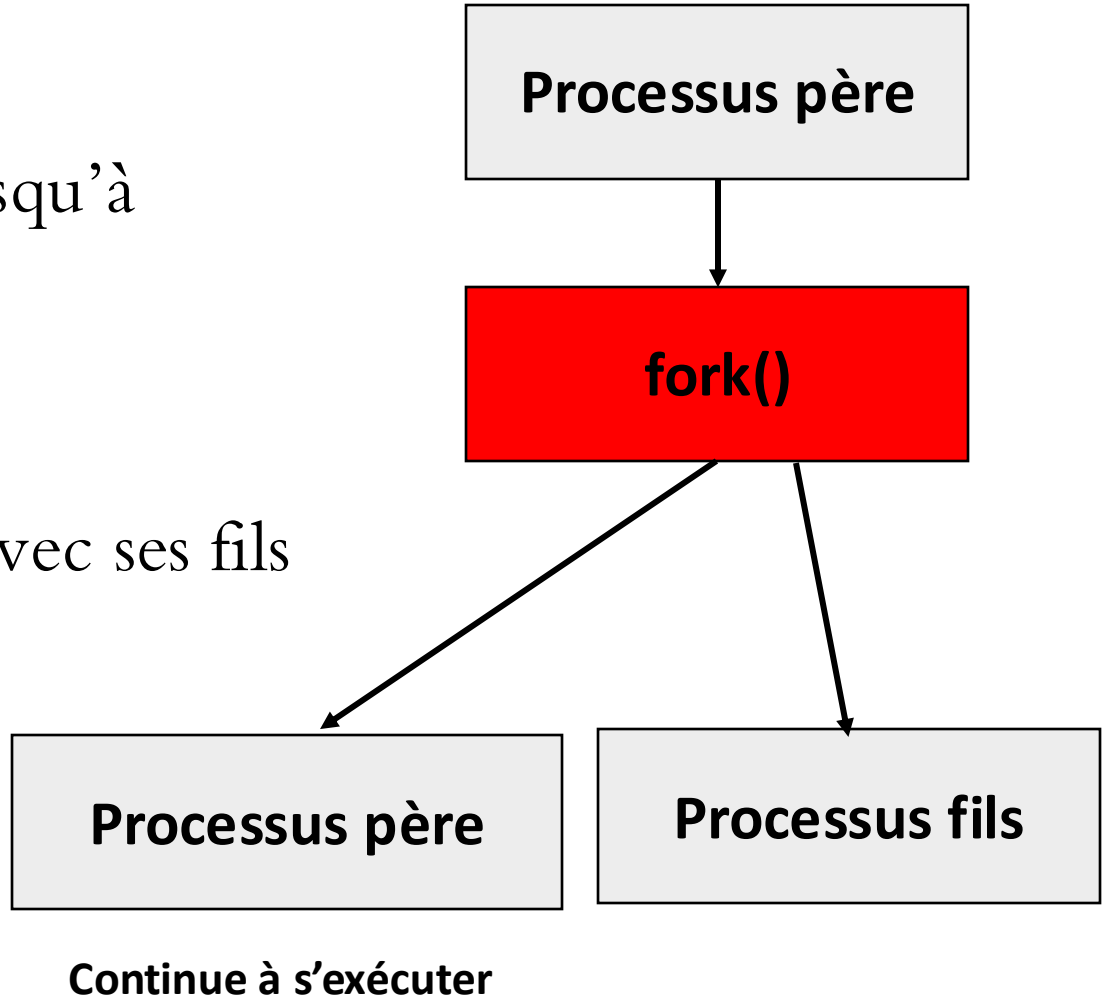
L’appel système fork()

L’exécution du processus père est suspendue jusqu’à terminaison du processus fils

–> Exécution séquentielle , SE Monotâche

Le père continue à s’exécuter en concurrence avec ses fils

–> Exécution asynchrone , SE MultiTâche



L’appel système fork()

Le code retour du fork() est le seul moyen de différencier le père du fils.

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

Définit un certain nombre de types de données de base utilisés dans les autres en–têtes système.

pid_t : identificateur de processus

L’appel système fork()

Le code retour du fork() est le seul moyen de différencier le père du fils.

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

Bibliothèque standard en C et en C++ qui fournit l'accès à un ensemble de fonctions de bas niveau du système d'exploitation, principalement pour les opérations d'entrée/sortie (E/S) et le contrôle des processus.

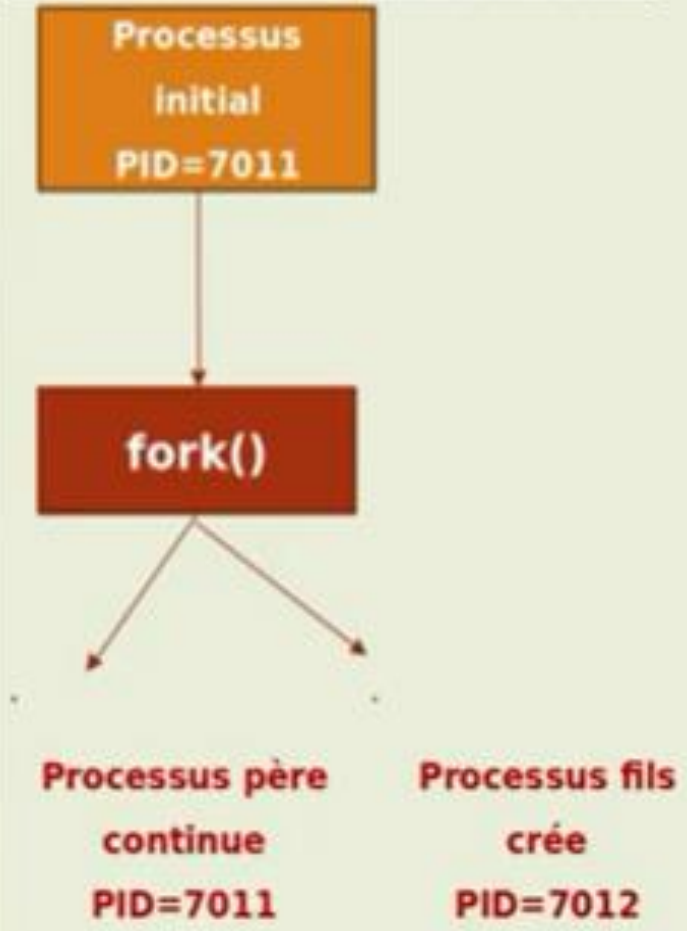
fork(), getpid(), exec(), sleep()

L’appel système fork() : Exemple 1

Exemple1.c

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Coucou %d \n« , getpid());
    fork(); /* création du processus fils */
    printf(« Mon   PID   %d:   PID   père
%d\n", getpid(), getppid());
}
```

Schéma des processus



Résultats

Coucou 7011

Mon PID 7012: PID père 7011

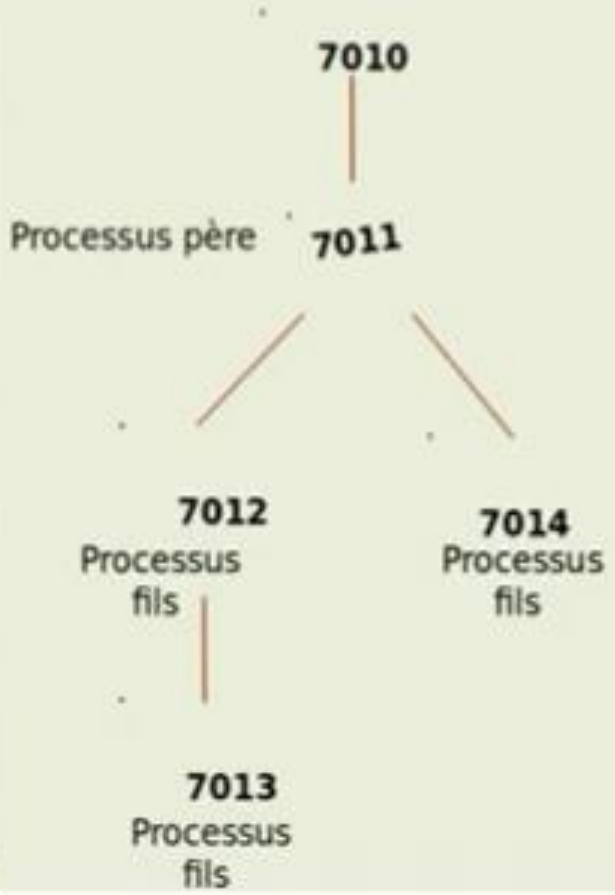
Mon PID 7011: PID père 5668

L’appel système fork() : Exemple 2

Exemple2.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("Coucou %d \n« , getpid());
    fork(); /* création du processus fils */
    fork(); /* création du processus fils */
    printf(« Mon   PID   %d:   PID   père
%d\n", getpid(), getppid());
}
```

Arbre binaire des processus



Résultats

Coucou **7011**

Mon PID **7011**: PID père **7010**

Mon PID **7012**: PID père **7011**

Mon PID **7014**: PID père **7011**

Mon PID **7013**: PID père **7012**

L’appel système fork() : Exemple 3

Exemple3.c (avec la structure if)

Processus père

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    pid = fork();
    if (pid ==-1)
        printf("Echec de création\n");
    else if (pid==0)
        printf("Hello Dad\n");
     else
        printf("Bonjour fils\n");
}
```

Processus fils

```
if (pid ==-1)
    printf("Echec de création\n");
else if (pid==0)
    printf("Hello Dad\n");
 else
```

Résultats

Bonjour fils

Hello Dad

L’appel système fork() : Exemple 4

Exemple4.c (avec la structure switch)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    int pid;
    pid = fork(); /* création du processus fils */
    switch(pid)
    {
    case (-1) : printf("Echec de création.");
                exit(-1);
                break;
    case(0) : printf("Je suis le fils, PID %d",getpid());
              break;
    default : printf("Je suis le père, PID %d", getpid());
    }
}
```

Résultats

Je suis le fils, PID 7011

Je suis le père, PID 7010

L’appel système fork() : Exemple 5

Exemple5.c: Partage de variables

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(){
    pid_t pid;
    int i=5,j=2;
    pid = fork();
    if (pid ==-1)
        printf("Echec de création\n");
    else if (pid==0){ /*processus fils*/
         printf("Je suis le fils, PID %d\n",getpid());
        j--;}
    else{ /*processus père */
         printf("Je suis le père, PID %d\n", getpid());
        i++;}
    printf("PID:%d i:%d j:%d\n", getpid(),i,j);
}
```

Résultats

Je suis le fils, PID 6774

PID: 6774 i:5 j:1

Je suis le père, PID 6773

PID:6773 i:6 j:2

L'appel système fork() : Exemple 6

```c
#include <sys/types.h>   // pour le type pid_t
#include <unistd.h>      // pour fork
#include <stdio.h>       // pour perror, printf

int main(void)
{
    pid_t p ;
    int   a = 20;

    // écration d'un fils
    switch (p = fork())
    {
        case -1:
            // le fork a echoue
            perror("le fork a echoue !") ;
            break;
        case 0 :
            // Il s'agit du processus fils
            printf("ici processus fils, le PID %d.\n", getpid());
            a += 10;
            break;
        default :
            // Il s'agit du processus pere
            printf("ici processus pere, le PID %d.\n", getpid());
            a += 100;
    }
    // les deux processus executent cette instruction
    printf("Fin du processus %d avec a = %d.\n", getpid(), a);
```

```
ici processus pere, le pid 12339.
ici processus fils, le pid 12340.
Fin du Process 12340 avec a = 30.
Fin du Process 12339 avec a = 120.
```

Terminaison d'un processus

Un processus peut se terminer **normalement** ou **anormalement**.

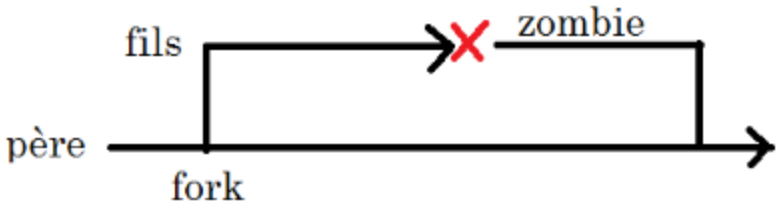
Dans le premier cas, l'application est abandonnée a la demande de l'utilisateur, où la tâche a accomplir est finie.

Dans le second cas, un dysfonctionnement est découvert, qui est si sérieux qu'il ne permet pas au programme de continuer son travail.

Orphelin et Zombie

- **Processus orphelins**
  - si un processus père meurt avant son fils ce dernier devient orphelin.

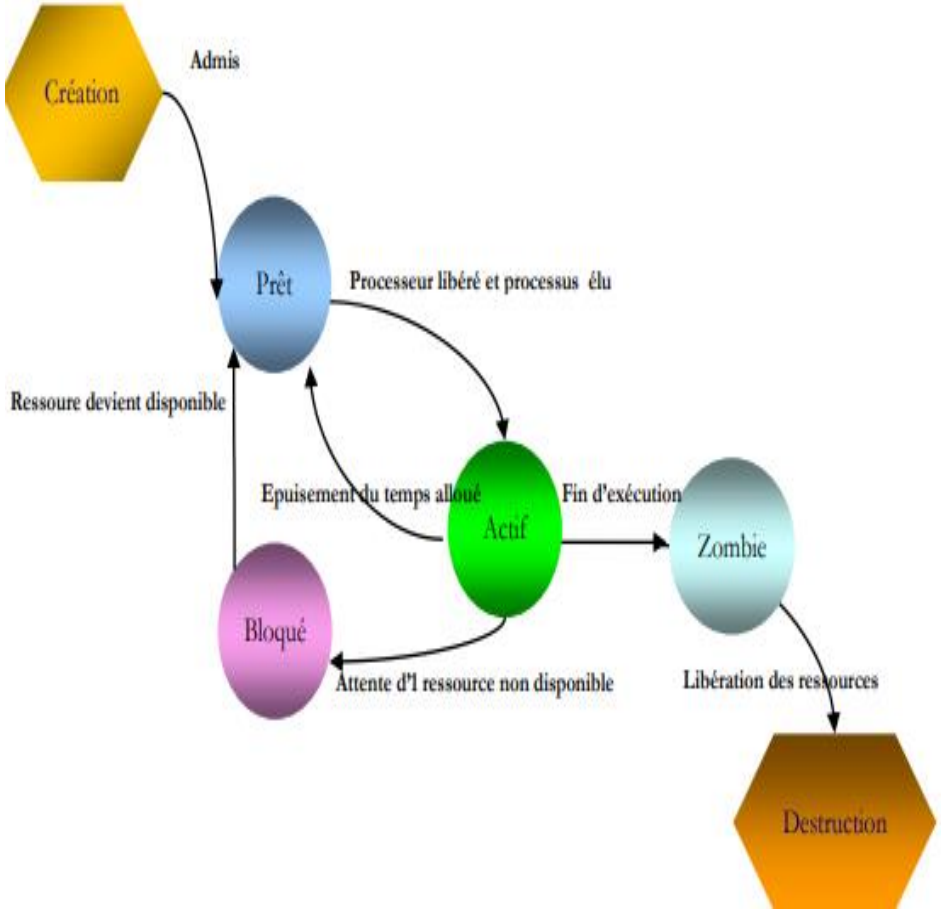
- **Processus zombie**
  - Si un fils se termine tout en disposant toujours d'un PID celui-ci devient un processus zombie
  - Le cas le plus fréquent : le processus s'est terminé mais son père n'a pas (encore) lu son code de retour.



Orphelin et Zombie

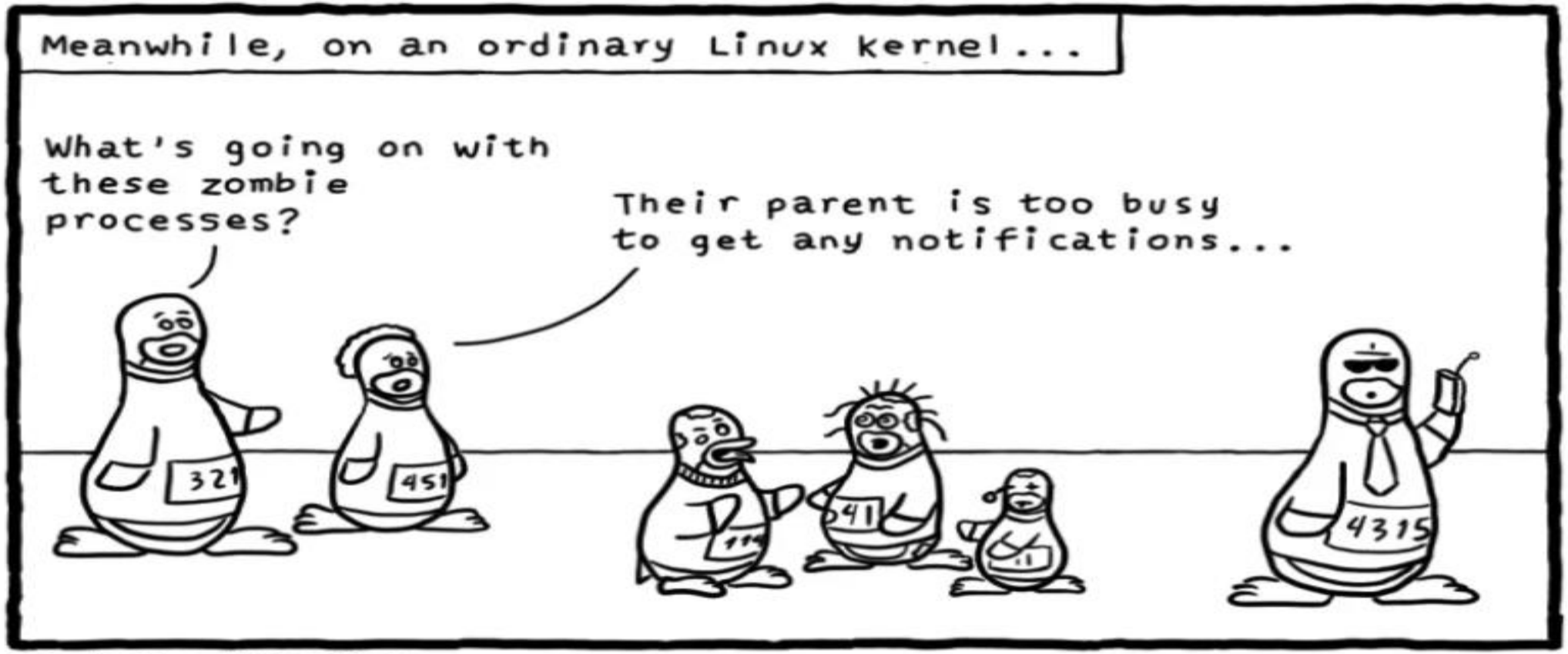
- Un processus fils peut devenir orphelin si son père termine avant lui, auquel cas le noyau s'arrange pour le « faire adopter » par un processus système (INIT), le processus fils peut donc lui transmettre son statut de terminaison.

- Un processus est dit zombie s'il s'est achevé, mais qui dispose toujours d'un identifiant de processus (PID) et reste donc encore visible dans la table des processus. On parle aussi de processus défunt.



Processus Zombie sous Linux

Les processus zombies ont un état "Z" dans la sortie de la commande ps aux. Le PID de ces processus est également affiché, ainsi que le PPID qui est normalement celui du processus parent qui a créé le processus zombie. Le nombre de processus zombies peut être obtenu en comptant le nombre de lignes qui affichent l'état "Z".



Processus Zombie sous Windows

Sous Windows, on peut utiliser le Gestionnaire des tâches pour identifier les processus zombies.

Dans le Gestionnaire des tâches, con clique sur l'onglet "Détails" pour afficher la liste des processus en cours d'exécution.
Les processus zombies apparaîtront avec un état "Suspendu" et un code de sortie de "-".

On peut trier la liste des processus par état en cliquant sur l'en-tête de la colonne "État", ce qui vous permettra de trouver facilement les processus zombies.