LINUX – LE NOYAU
LES PROCESSUS

Processus (ou tâche)

« Activité résultant de l'exécution d'un programme séquentiel, avec ses données, par un processeur »

- Ensemble d'instructions à exécuter (programme)
- Instance du programme à un instant T
- Environnement :
  - Fichiers ouverts,
  - Mémoire utilisée
  - Utilisation processeur
  - Etc…

« Multi-Tâches »

- Partage du ou des processeurs
- Alternance rapide d'exécution des processus présents en mémoire

L'un des premiers ordinateurs multitâche au monde conçu par Bull en 1958 (Gamma 60)

Intérêts du « Multi-Tâches »

Faire plusieurs activités en "même temps"

- Exemple :
  - Faire travailler plusieurs utilisateurs sur la même machine
    Chaque utilisateur a l'impression d'avoir la machine pour lui tout seul.
  - Compiler tout en lisant son mail

- Problème:
  - Un processeur ne peut exécuter qu'une seule instruction à la fois
- Objectif :
  - **Partager un (ou plusieurs) processeur** entre différents **processus**

Déclenchement des taches

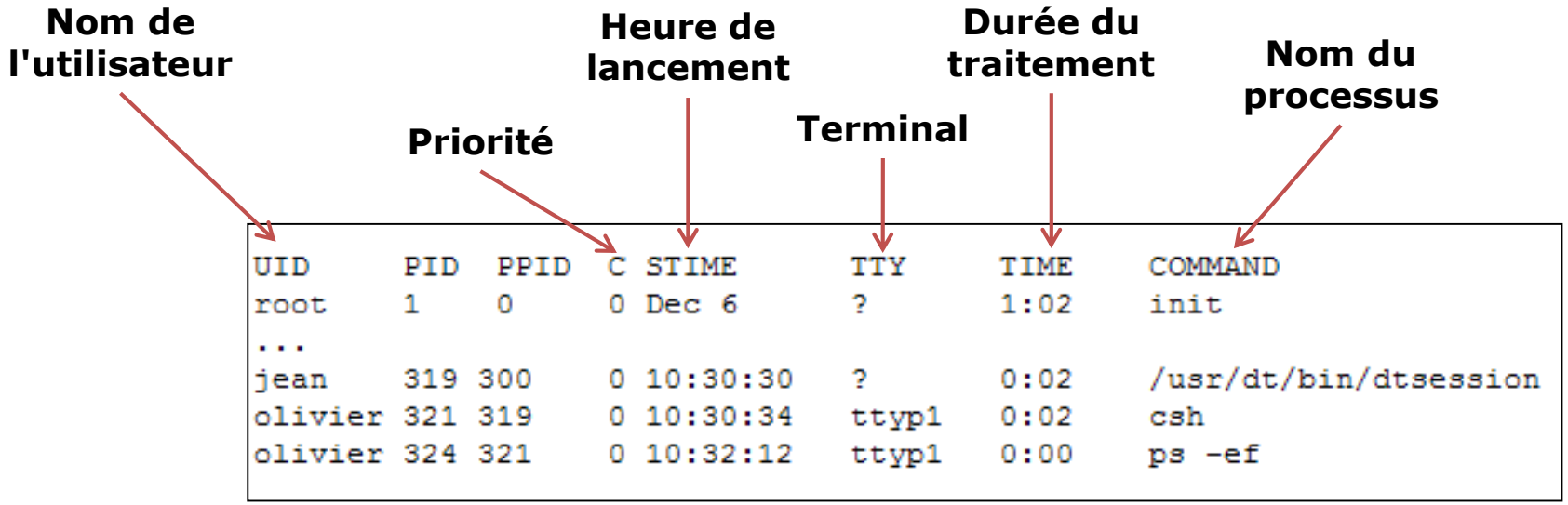
- Par interruption
  - **Tâche immédiate**

- Par une autre tâche
  - **Tâche différée**

Le processus créateur est appelé processus "père" ou "parent"
Le processus crée est appelé processus "fils"

Caractéristiques d'un processus

- Le système doit pouvoir gérer plusieurs processus en cours à un instant donné
- Nécessité d'identifier chacun des processus
  - Numéro unique appelé **PID** (Process IDentifier)
  - PID du processus père : **PPID** (Parent Process IDentifier)
  - Le premier processus lancé sur le système qui n'a pas de père et a pour PID 1



Etats des processus

- Au fur et a mesure qu'un processus s'exécute, il est caractérisé par **un état** :
  - Lorsque le processus obtient le processeur et s'exécute, il est dans l'état élu.
    L'état élu est l'état d'exécution du processus

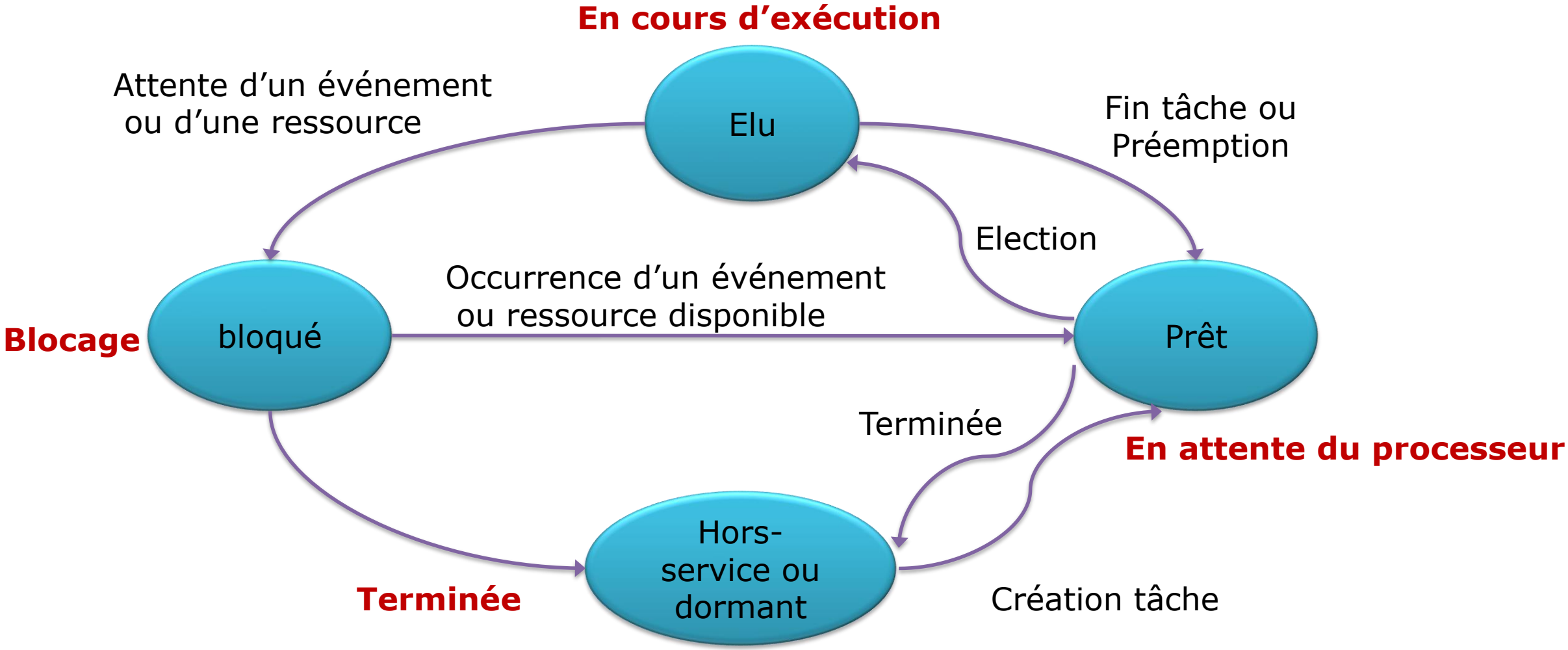
  - Lors de l'exécution, le processus peut demander à accéder à une ressource. Il quitte alors le processeur et passe dans l'état bloqué.
    L'état bloqué est l'état d'attente d'une ressource autre que le processeur

Gestion des taches

- Lorsque le processus est passé dans l'état bloqué, le processeur a été alloué à un autre processus. Le processeur n'est donc pas forcément libre. Le processus passe dans l'état prêt. **L'état prêt est l'état d'attente du processeur.**

- Le processus a terminé son exécution

Diagramme d'états



Bloc de contrôle de processus (PCB)

- PCB = Process Control Block
  C'est une structure de description du processus associé au programme exécutable

- Le PCB permet la sauvegarde et la restauration du contexte mémoire et du contexte processeur lors des opérations de commutations de contexte

Identificateur processus
État du processus
Compteur ordinal Contexte pour reprise (registres et pointeurs, piles …)
Chaînage selon les files de l'Ordonnanceur Priorité (ordonnancement)
Informations mémoire (limites et tables pages/segments)
Informations sur les ressources utilisées fichiers ouverts, outils de synchronisation, entrées-sorties
Informations de comptabilisation

Création de processus

- Un processus peut créer un ou plusieurs autres processus en invoquant un appel système de création de processus
  - Le processus créateur -> processus père
  - Les processus créés -> processus fils
  - Développement d'un arbre de filiation entre processus

- Différentes variantes :
  - Le processus créé hérite ou non de données et du contexte de son processus créateur
  - Le processus créé peut s'exécuter parallèlement à son père.
  - Dans certains systèmes, le processus père doit attendre la terminaison de ses fils pour pouvoir reprendre sa propre exécution

Destruction de processus

- Plusieurs possibilités :
  - Le processus a terminé son exécution
    - Le processus s'autodétruit en appelant une routine système de fin d'exécution
  - Le processus commet une erreur irrécouvrable
    - Le processus est terminé par le système
  - Les autres processus demandent la destruction du processus
    - Appel à une routine système

- Le contexte du processus est démantelé
  - Les ressources allouées au processus sont libérées
  - Le bloc de contrôle est détruit

Suspension d'exécution

- Momentanément arrêter l'exécution d'un processus pour la reprendre ultérieurement

  Le contexte du processus est sauvegardé dans son PCB

  Le processus passe dans l'état bloqué

- Reprise d'exécution

  Transition de déblocage, le processus entre dans l'état prêt

Affichage du numéro d'un processus

```
#include <unistd.h>
pid_t getpid()
```

Retourne :
- **Le numéro du processus courant (PID)**

```
pid_t getppid()
```

Retourne :
- **Le numéro du processus parent du processus courant (Parent PID)**

Création de processus : system()

```
int system(cmd)
        char *cmd;
```

Retourne :
- **127 si l'appel système pour /bin/sh échoue**
- **-1 si une autre erreur se produit**
- **ou le code de retour de la commande sinon**

Exécute la commande indiquée dans **cmd** en appelant **/bin/sh -c cmd** et revient après l'exécution complète de la commande

Exemple

```
main()
{
        printf("Début exemple...\n");
        system("ps -aux");
        printf("Retour...\n");
}
```

Duplication de processus : fork()

Créer un processus fils (child)

```
#include <unistd.h>
pid_t fork( );
```

En cas de succès, le PID du fils est renvoyé au processus parent et 0 est renvoyé au processus fils

En cas d'échec -1 est renvoyé dans le contexte du parent aucun processus fils n'est créé

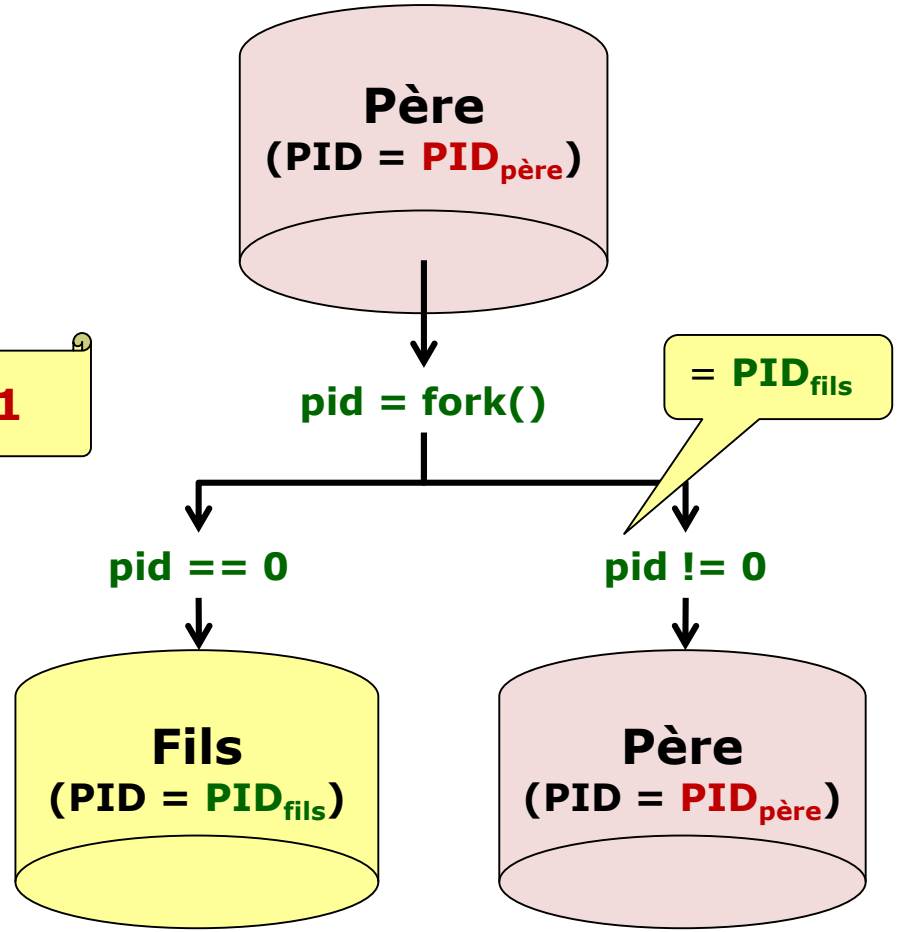
La fonction fork()

fork() lance un nouveau processus qui est la copie conforme du processus courant

La valeur prise par la fonction **fork()** dans l'ancien processus (le père) correspond au numéro d'identification du nouveau processus (le fils) alors que la valeur transmise dans le fils est 0

En général on a $PID_{fils} = PID_{père} + 1$

Les espaces de données sont complètement séparés => la modification d'une variable dans l'un est invisible dans l'autre



Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
    pid_t pidFork;
    quisuisje = "le père";
    pidFork = fork();
    if (pidFork == 0) {
        quisuisje = "le fils";
        printf("je suis %s", quisuisje);
    }
    else {
        printf("je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

```
#./a.out
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
        pid_t pidFork;
        quisuisje = "le père";
        pidFork = fork();
        if (pidFork == 0) {
                quisuisje = "le fils";
                printf("je suis %s", quisuisje);
        }
        else {
                printf("je suis %s", quisuisje);
                wait(NULL);
        }
        return 0;
}
```

```
#./a.out
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 1000
      if (pidFork == 0) {
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else {
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

Le Fils

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 0
      if (pidFork == 0) {
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else {
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

```
#./a.out
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 1000
      if (pidFork == 0) {
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else {
            printf("je suis %s",quisuisje);
            wait(NULL);
      }
      return 0;
}
```

Le Fils

Descripteurs de fichiers
1: (stdout) position = 0

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 0
      if (pidFork == 0) {
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else {
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

```
#./a.out
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 15

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 1000
      if (pidFork == 0){
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else{
            printf("je suis %s", quisuisje);
      ➡ wait(NULL);
      }
      return 0;
}
```

Le Fils

Descripteurs de fichiers
1: (stdout) position = 15

Variables globales
quisuisje = "le fils"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 0
      if (pidFork == 0){
            quisuisje = "le fils";
      ➡ printf("je suis %s", quisuisje);
      }
      else{
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

```
#./a.out
je suis le père
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 30

Variables globales
quisuisje = "le père"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 1000
      if (pidFork == 0){
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else{
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

Le Fils

Descripteurs de fichiers
1: (stdout) position = 30

Variables globales
quisuisje = "le fils"

```
main() {
      pid_t pidFork;
      quisuisje = "le père";
      pidFork = fork(); // pidFork <- 0
      if (pidFork == 0){
            quisuisje = "le fils";
            printf("je suis %s", quisuisje);
      }
      else{
            printf("je suis %s", quisuisje);
            wait(NULL);
      }
      return 0;
}
```

```
#./a.out
je suis le père
je suis le fils
```

Exemple d'utilisation de fork()

Le Père

Descripteurs de fichiers
1: (stdout) position = 30

Variables globales
quisuisje = "le père"

```
main() {
        pid_t pidFork;
        quisuisje = "le père";
        pidFork = fork(); // pidFork <- 1000
        if (pidFork == 0){
                quisuisje = "le fils";
                printf("je suis %s", quisuisje);
        }
        else{
                printf("je suis %s", quisuisje);
                wait(NULL);
        }
        return 0;
}
```

```
#./a.out
je suis le père
je suis le fils
```

Exemple d'utilisation de fork()

Attente et fin d'un processus

```
#include <sys/types.h>
#include <sys/wait.h>

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

Suspend l'exécution du processus courant jusqu'à ce qu'un enfant se termine ou jusqu'à ce qu'un signal à intercepter arrive

En cas de réussite, le PID du fils qui s'est terminé est renvoyé, en cas d'échec -1 est renvoyé !

```
#include <stdlib.h>

void exit(status)
        int status;
```

C'est la manière normale de terminer un processus
Tous les descripteurs de fichiers sont fermés

Les fonctions exec

execl(), execlp(), execle(), execv(), execvp()

Ces fonctions remplacent le programme en cours d'exécution par un autre !

```
int execl(char *path, char *arg, ...);
int execlp(char *file, char *arg, ...);
int execle(char *path, char *arg, ..., char *envp[]);
int execv(char *path, char *argv[]);
int execvp(char *file, char *argv[]);
```

Lettre p -> Recherche du programme en suivant le PATH courant (execlp, execvp)

Lettre v -> Liste d'arguments sous forme d'un tableau de pointeur (execv, execvp)

Lettre e -> un argument supplémentaire : tableau de variables d'environnement (execle)

Exemple d'utilisation de execl()

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

int main() {
  pid_t pidFork;
  pidFork = fork();
  if (pidFork == 0) {
    printf("je suis le fils... Je dors 5 secondes puis execute la commande date\n");
    sleep(5);
    execl("/bin/date", "date", NULL);
    /* on sort de execl seulement si une erreur se produit */
    perror("execl() échec!\n");
    exit(-1);
  }
  else {
    printf("je suis le père ...\n");
    wait(NULL);
  }
  exit(0);
}
```

Exemple d'utilisation de execl()