



# Systemes d'exploitation

Processus

# Linus Torvalds

---



- Finlandais
- Étudiant à Helsinki
- Git
- Linux



- Processus
  - Rôle
  - Attributs
- Hiérarchie de processus
- Créer un processus
  - Autres opérations de traitement
- Fermer un processus



# Bibliographie pour aujourd'hui

---

- Modern Operating Systems
  - Chapitre 2
    - 2.1
- Operating Systems Concepts
  - Chapitre 3

# Processus

---

- le unité de base du exécution
- en SE, il y a deux abstractions de base
  - **fichier** - abstraction pour l'information / les données
  - **processus** – abstraction pour des actions

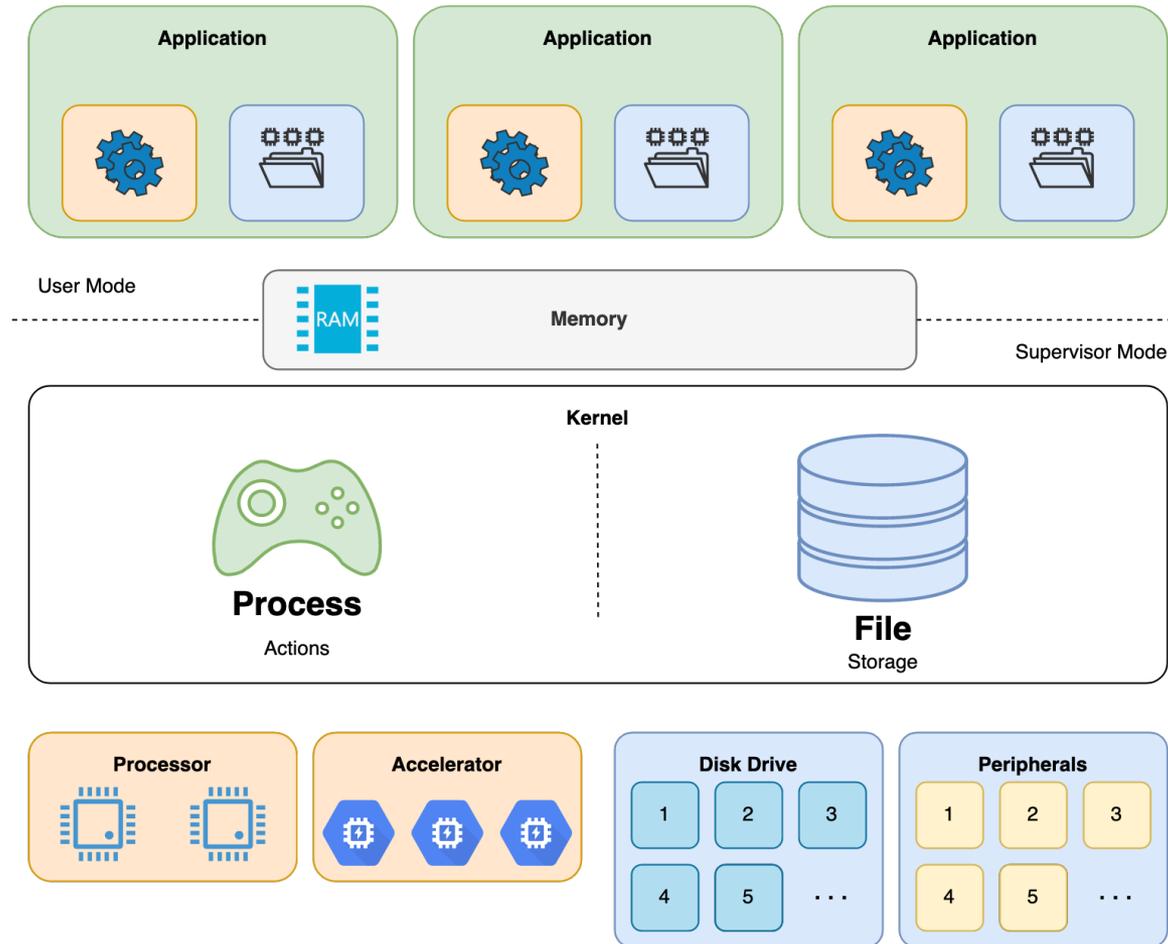
# PROCESSUS

# Qu'est-ce qu'un processus?

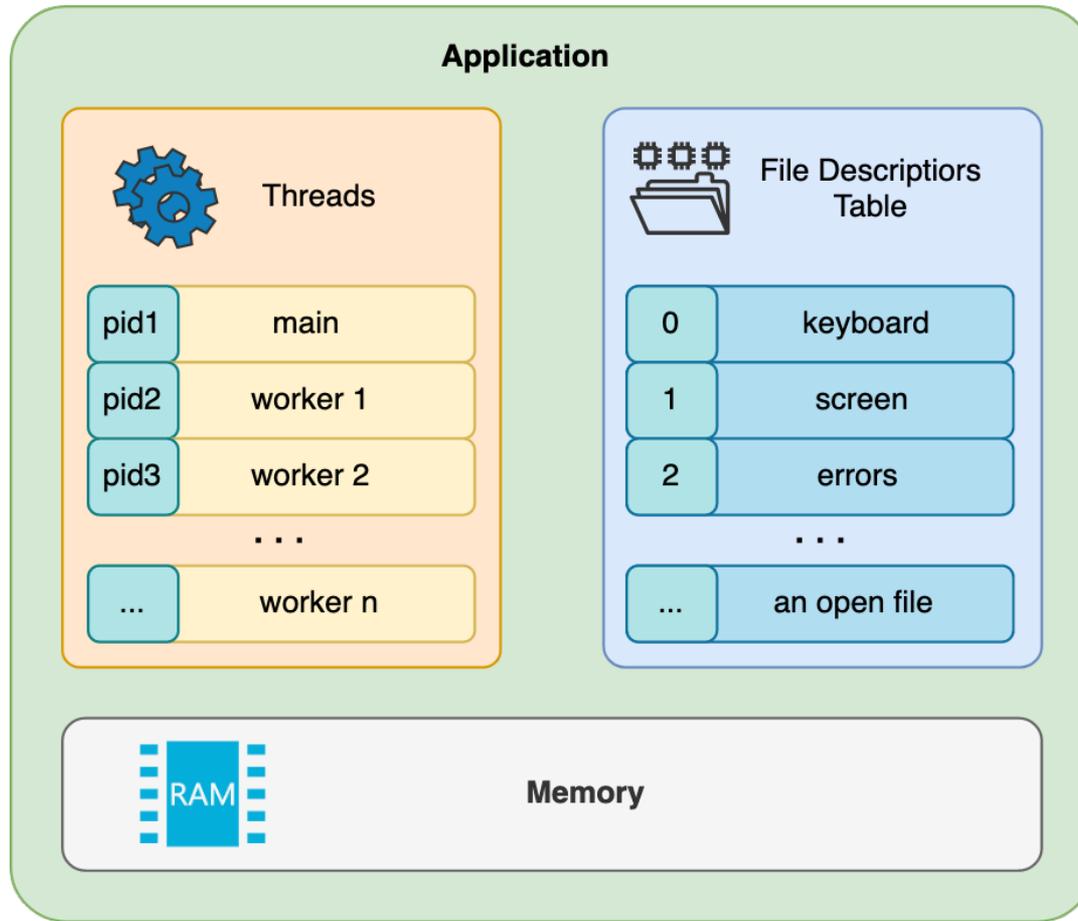
---

- Un programme en cours d'exécution
- Encapsulation / abstraction de l'exécution dans SO
- Abstraction sur processeur, mémoire, E / S

# Abstractions - Idée General



# API utilisée par les processus



# Processeur

---

- Un processus a un ou plusieurs threads
  - Un thread exécute des instructions sur un processeur
- En règle générale, les processus système sont plus que des processeurs système
  - La planification des processus sur les processeurs est nécessaire

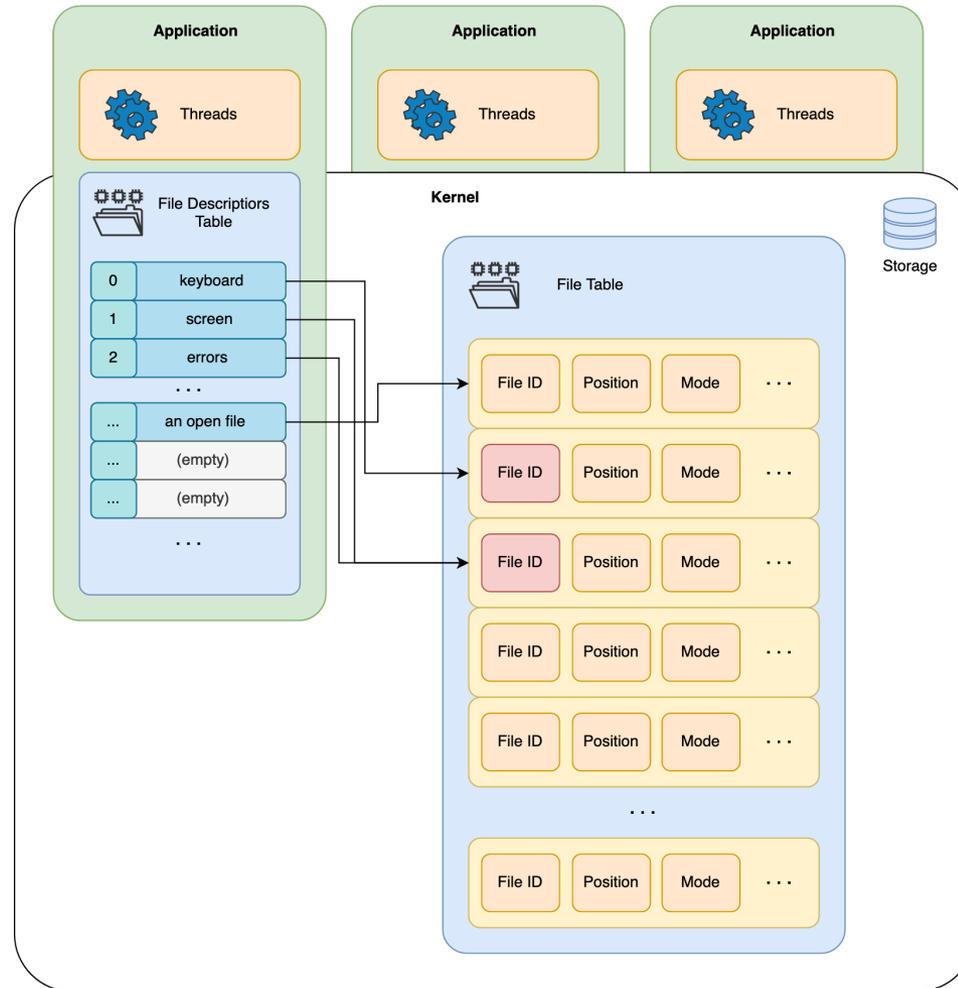
# Memoire

---

- Un processus a sa propre mémoire (isolée des autres processus)
  - Code
  - Instructions
  - Données
- Les instructions sont amenées de la RAM dans le processeur et exécutées
  - Nous disons que le processus s'exécute sur le processeur

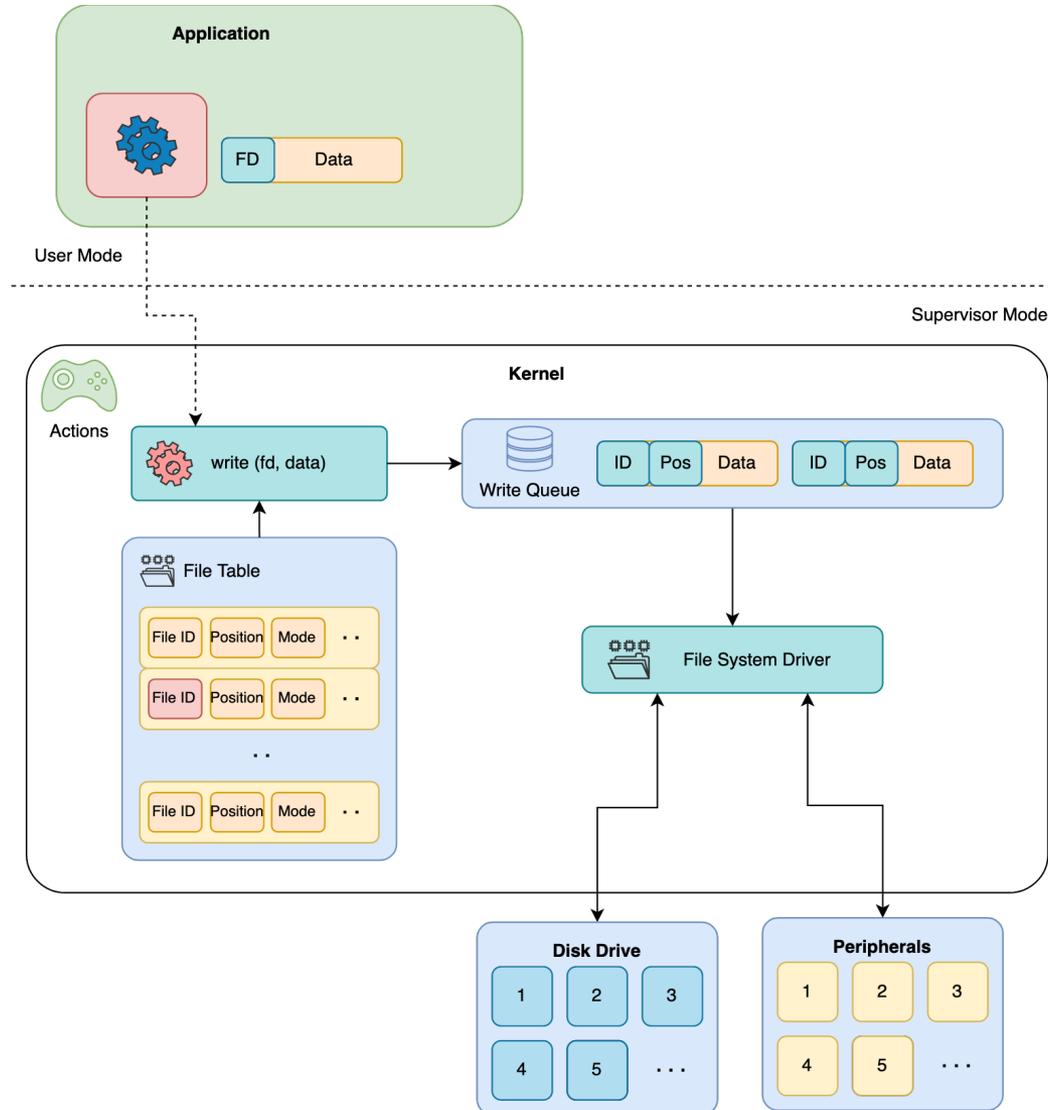
- Un processus communique avec l'extérieur: disque, réseau, clavier, moniteur
- La communication se fait généralement par les descripteurs de fichiers.
  - tableau de descripteurs de fichier
  - descripteur de fichier: un fichier, un socket, un terminal, un périphérique, etc.
- **Les opérations d'E/S bloque le processus**
  - E/S est plus lent que le processeur
  - le processus attend la fin de l'opération

# Tableau des fichiers



L'interface E/S de processus

# Appel du système bloquante



# Types de processus (CPU)

---

- CPU bound (CPU intensive)
  - utilise beaucoup le processeur
- I/O bound (I/O intensive)
  - utilise rarement le processeur
  - faire des opérations d'E / S -> se bloque

# Types de processus (Utilisateur)

---

- Interactif (foreground)
  - interagir avec l'utilisateur
- Non interactif (batch, background)
  - services, démons

# LES ATTRIBUTES D'UN PROCESSUS

# A quoi ressemble un processus SE?

---

- **PID** (Process ID)
  - le identifiant de processus
- **PCB** (Process Control Block)
  - Une structure de données
  - Décrit un processus au niveau SO
- Informations sur les ressources utilisées
- Liens vers d'autres structures
- Sécurité, surveillance, informations comptables

# Ressources

---

- Temps d'exécution sur processeur
- Mémoire (code et données)
- Tableau des descripteurs de fichiers
  
- Certaines ressources peuvent être partagées avec d'autres processus

# Attributs d'un processus

---

- **PID**
- parent PID
- pointeurs vers les ressources
  - tableau de descripteurs, tableau de mémoire ...
- **état** (en cours, en attente)
- le **quantum** de temps d'exécution
- comptabilisation des ressources consommées
- **utilisateur**, groupe

# CREATION D'UN PROCESSUS

# Nouveau processus

---

- À partir d'un exécutable (programme)
- Un autre processus (parent) crée un processus (enfant)
- Le nouveau processus (processus enfant) remplit sa mémoire avec des informations exécutables
  - L'action s'appelle également load, load time
  - fait par le chargeur (loader)

# Hiérarchie des processus

---

- Un processus peut créer un ou plusieurs processus enfants.
- Un processus peut avoir un processus parent

# Démarrage - UNIX

---

- *init* est au sommet de la hiérarchie des processus
  - PID 1
- *init* est créé par le noyau au boot
- *init* crée ensuite les processus de démarrage
- les processus de démarrage créent d'autres processus, etc.

# Processus et exécutables

---

- **Un ou plusieurs processus** proviennent **d'un fichier exécutable**
- Le fichier exécutable contient essentiellement les données et le code du futur processus
- Le processus a aussi des zones de mémoire non décrites dans l'exécutable associé
  - stack
  - heap (malloc)
  - zones pour bibliothèques dynamiques

# Nouveau Processus - UNIX

---

- Séparation entre la **création d'un nouveau processus** et le **chargement de données à partir d'un exécutable**
- **fork**: créer un nouveau processus (enfant) (presque identique au processus parent)
- **exec**: chargement d'informations d'un exécutable dans la mémoire du processus enfant

- Nouveau processus
- Une clone de processus parent
- **retour deux fois**
  - en parent
  - en enfant
  
- Le nouveau processus a
  - Un copie de PCB
  - Nouveau PID
  - Nouveau tableau des descripteurs (**avec les même pointeurs**)
  - Un copie de la mémoire

# fork bomb

---

```
while (1)
{
    fork ();
}
```

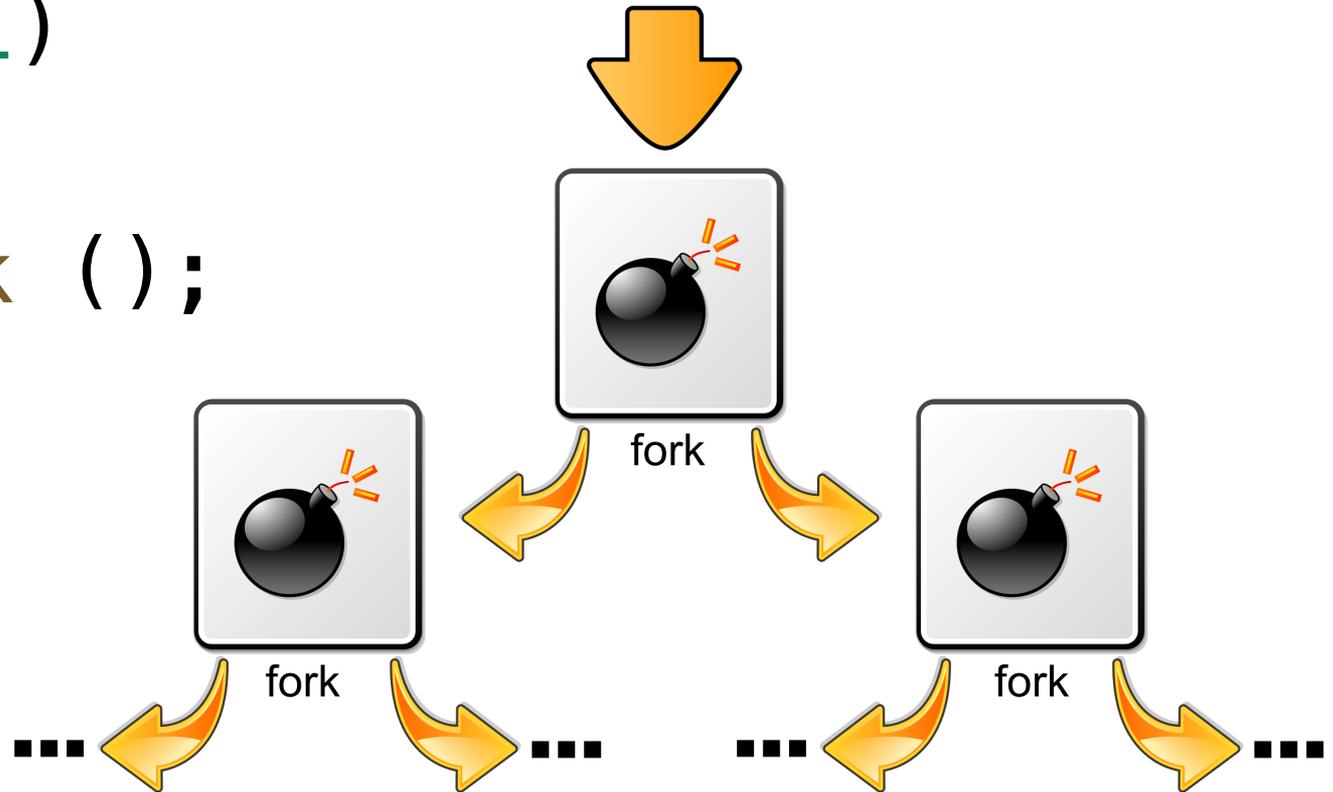
# fork bomb

```
while (1)
```

```
{
```

```
    fork ();
```

```
}
```



# Nouveau Processus - UNIX - fork

---

```
pid_t pid = fork ();
if (pid < 0) {
    perror ("fork");
}
else if (pid > 0) {
    // This is the parent process
    printf ("Child process PID is %d\n", pid);
}
else {
    // This is the child process
    printf ("My PID is %d\n", getpid ());
}
```

# exec

---

- Charge un nouveau exécutable
- Remplace les données d'exécutable courant (*l'image*) avec les données chargées d'exécutable
- **ne retour pas**

# Nouveau Processus - UNIX - exec

---

```
int execl(const char *path, const char *arg0, ...  
/*, (char *)0 */);
```

```
int execl(const char *path, const char *arg0, ...  
/*, (char *)0, char *const envp[] */);
```

```
int execv(const char *path, char *const argv[]);
```

```
int execve(const char *path, char *const argv[],  
char *const envp[]);
```

```
int execvp(const char *file, char *const argv[]);
```

# Nouveau Processus - UNIX - exec

---

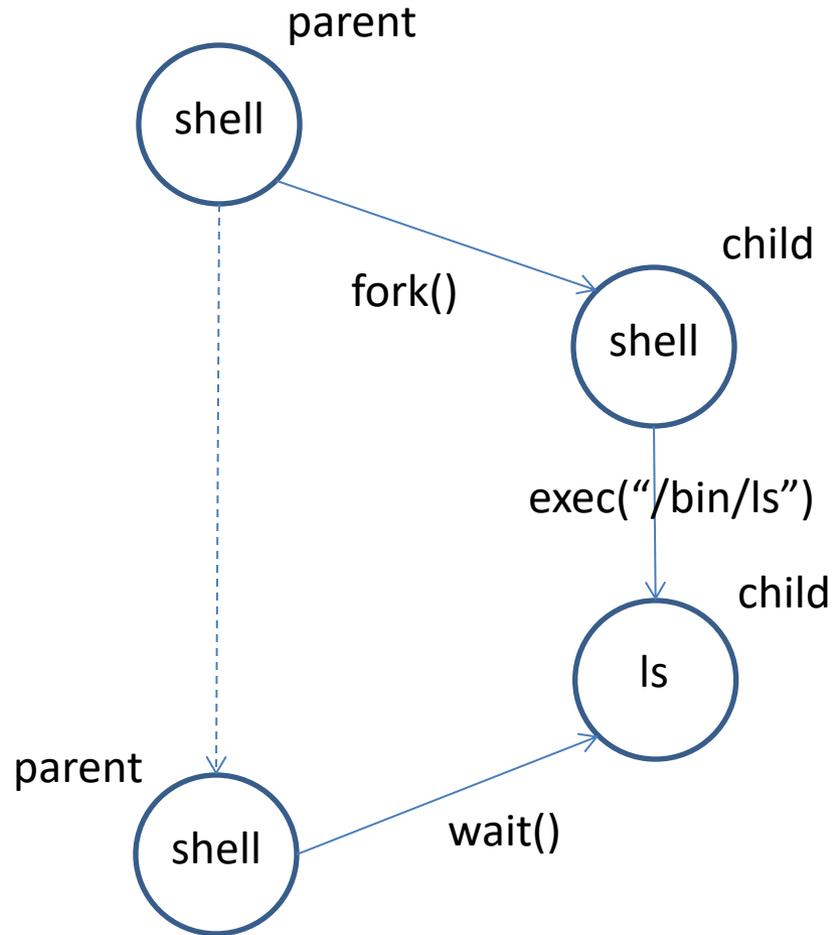
```
if (!execl ("/usr/bin/cowsay", "/usr/bin/cowsay",  
"Hello", NULL))  
{  
    perror ("execl");  
    abort ();  
}  
// if successful, this is never executed
```

# Exemple

---

1. Une chaîne est écrite au stdin
2. La chaîne est interprétée par le shell dans un chemin d'exécutable (et des arguments)
3. Le processus shell crée un nouveau processus (**fork**)
  - Le processus enfant est planifié par le planificateur
4. Le processus enfant "charge" les données et le code de l'exécutable (**exec**)
  - Le processus parent du processus enfant est le processus shell
5. Le processus parent attend le fini du processus enfant

# Exemple



# Exemple

---

```
int status;
pid_t pid = fork ();
if (pid < 0) {
    perror ("fork");
    abort ();
}
else
if (pid > 0) {
    // This is the parent process
    // printf ("Child process PID is %d\n", pid);
    waitpid (pid, &status, 0);
}
else {
    // This is the child process
    if (!execl ("/bin/ls", "/bin/ls", "-l", NULL)) {
        perror ("Error loading /bin/ls\n");
        abort ();
    }
}
}
```

# fork, exec et redirection

---

```
int fd;
pid_t pid = fork ();
if (pid == 0)
{
    // redirect
    fd = open ("output", O_TRUNC | O_WRONLY | O_CREAT, 0644);
    dup2 (fd, 1);
    if (!execl ("/bin/ls", "/bin/ls", "-l", NULL))
    {
        perror ("Error loading /bin/ls\n");
        abort ();
    }
}
```

- Processus
- init
- Hiérarchie des processus
- fork
- exec

# Questions

---

