

# Système d'Exploitation

## Introduction aux SE Programmation de processus

Juan Angel Lorenzo del Castillo

Contributions de : Thierry Garcia, Florent Devin et Taisa Guidini Gonçalves  
et Mariem ALLOUCH MAHDI

2025–2026



# Plan

1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Introduction

# Motivation

- **Système d'exploitation** : lien entre le matériel, le logiciel et l'utilisateur.
  - Comment un programme (i.e. du logiciel) peut être transposé au plan physique ?
  - Comment utiliser un ordinateur de façon efficace ?
- Discipline fondamentale et transversale pour l'Ingénierie Informatique.
- Essentielle pour comprendre d'autres matières :
  - ▶ Architecture des Ordinateurs (ING1)
  - ▶ Programmation Système et Réseau (ING2)
  - ▶ Architecture et Programmation Parallèle et Distribuée (ING2)
  - ▶ Architecture réseau (ING2)
  - ▶ Projets ING2
  - ▶ etc.

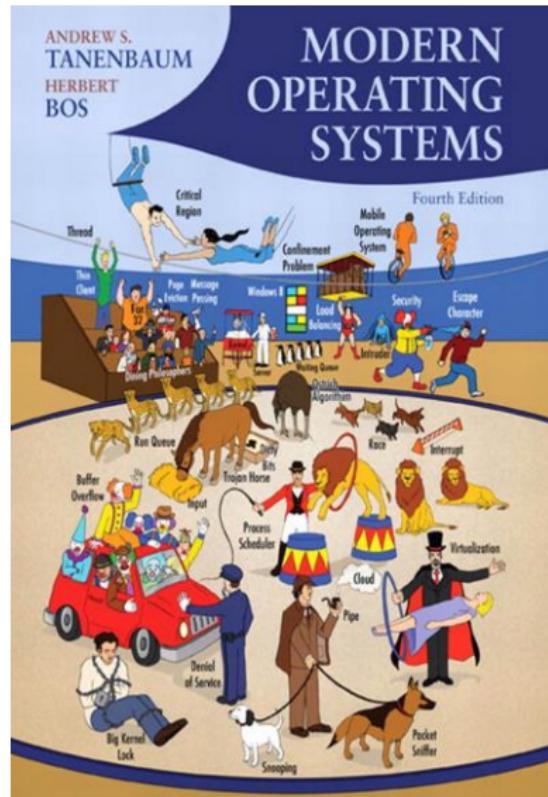
# Objectifs

- ❑ Connaître les fonctionnalités des SE
  - Structure
  - Gestion des ressources matérielles
  - Administration des programmes
  - Interface avec l'utilisateur
- ❑ Du point de vue de l'utilisateur, de l'administrateur et aussi du designer.
- ❑ En mettant l'accent sur [Linux/Unix](#).

# Thèmes à aborder

- ❑ Processus et cycle de vie
- ❑ L'ordonnanceur d'un SE
- ❑ Gestion de la mémoire
- ❑ Système des fichiers
- ❑ Processus de démarrage d'un SE
- ❑ Installation et configuration d'un SE
- ❑ Docker

# Bibliographie



# Évaluation

- Examen final papier
- 2 heures
- Calculatrice autorisée

# Plan

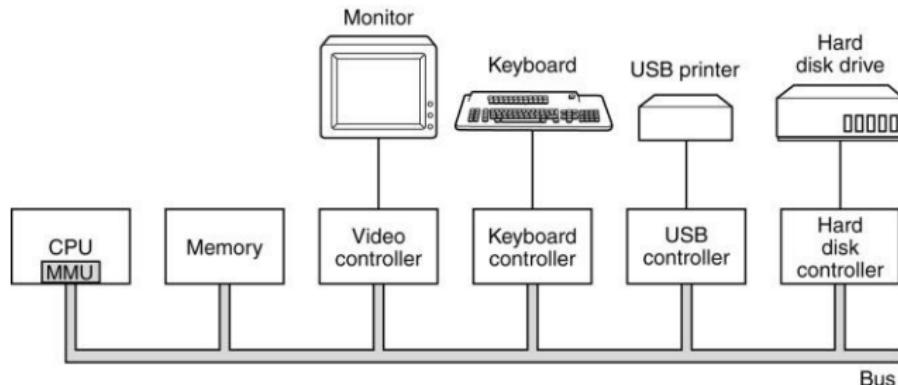
1. Introduction
2. **Le Système d'Exploitation**
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Le Système d'Exploitation

# L'ordinateur moderne, un système complexe

## ❑ Composants :

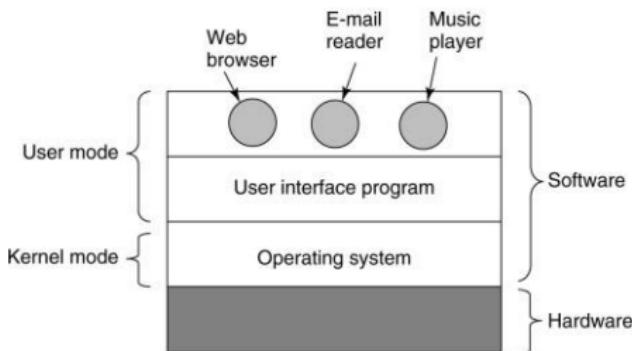
- Un ou plusieurs microprocesseurs
- Mémoires
- Disques
- Clavier
- Souris
- Displays (écran)
- Interfaces réseau
- D'autres périphériques d'entrées / sorties
- Etc.



# L'ordinateur moderne, un système complexe

## □ Composants :

- Programmes **systèmes** qui permettent le fonctionnement de l'ordinateur (**le système d'exploitation**)
  - \* Exécutés en **mode kernel** (aussi appelé mode *système* ou *superviseur*) : accès total au matériel et à toutes les instructions du processeur.
- Programmes d'**application** des utilisateurs
  - \* Exécutés en **mode utilisateur** : restreint l'accès aux ressources de l'ordinateur. Seulement un sous-ensemble des instructions du processeur sont disponibles (Exemple : instructions d'E/S interdites et accès mémoire protégé).
- Matériel (*Hardware*)



# L'ordinateur moderne, un système complexe

- ❑ **Problème :** L'administration et l'utilisation efficace deviennent difficile
  - Comment un programme peut accéder aux périphériques ?
  - Où à la mémoire (physiquement) ?
  - Comment gérer des erreurs ?
- ❑ Ces problèmes font partie du rôle d'un SE

## Système d'exploitation

Couche logiciel qui s'occupe de :

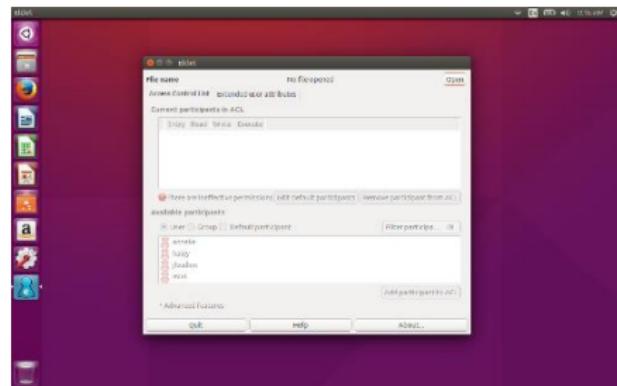
- fournir aux programmes d'application un modèle plus simple et propre de l'ordinateur (**présentation**).
- gérer l'utilisation de toutes les ressources de l'ordinateur (**gestion**).

# Fonctions du Système d'Exploitation

□ **Présentation** : Fournir un ensemble abstrait des ressources aux programmes/développeurs.

- Les programmes d'application interagissent avec le SE.
- Les utilisateurs interagissent avec l'interface utilisateur (Shell ou GUI - *Graphical User Interface*).
- Abstraction plus simple et plus agréable que le matériel.

```
overide@Atul-HP:~$ ls -l
total 212
drwxrwxr-x 5 override override 4096 May 19 03:45 acadenv
drwxrwxr-x 4 override override 4096 May 27 10:20 acadview_demo
drwxrwxr-x 12 override override 4096 May 3 15:11 anaconda3
drwxrwxr-x 2 override override 4096 Aug 1 13:49 apt
drwxrwxr-x 2 override override 4096 Oct 21 2016 Documents
drwxrwxr-x 7 override override 4096 Jun 1 13:09 Downloads
-rw-rw-r-- 1 override override 8980 Aug 8 2016 examples.desktop
-rw-rw-r-- 1 override override 45805 May 28 01:40 hs_err_pid1971.log
-rw-rw-r-- 1 override override 45147 Jul 1 03:24 hs_err_pid20006.log
drwxrwxr-x 2 override override 4096 Mar 2 10:22 Music
drwxrwxr-x 2 override override 4096 Sep 29 2016 newbin
drwxrwxr-x 5 override override 4096 Dec 20 22:44 nltk_data
drwxrwxr-x 4 override override 4096 May 31 20:46 Pictures
drwxrwxr-x 2 override override 4096 Aug 8 2016 Public
drwxrwxr-x 2 override override 4096 May 31 10:49 scripts
drwxrwxr-x 2 override override 4096 Aug 8 2016 Templates
drwxrwxr-x 2 override override 4096 Apr 14 13:21 Text
drwxrwxr-x 2 override override 4096 Mar 11 13:27 Videos
drwxrwxr-x 2 override override 4096 Sep 1 2016 xdm-helper
override@Atul-HP:~$
```



# Fonctions du Système d'Exploitation

- ❑ **Gestion** des ressources matérielles. Le SE doit connaître en détail le matériel de l'ordinateur.
  - Ordonne et contrôle l'allocation des ressources aux programmes.
  - Résolution des conflits entre programmes ou entre utilisateurs.
  - Gestion des entrées / sorties.
  - Gestion des processus (charger, exécuter, terminer).
  - Gestion de la mémoire centrale.
  - Utilisation partagée (*multiplexing*) : **en temps** (Ex. CPU - processeur) et **en space**(Ex. mémoire).

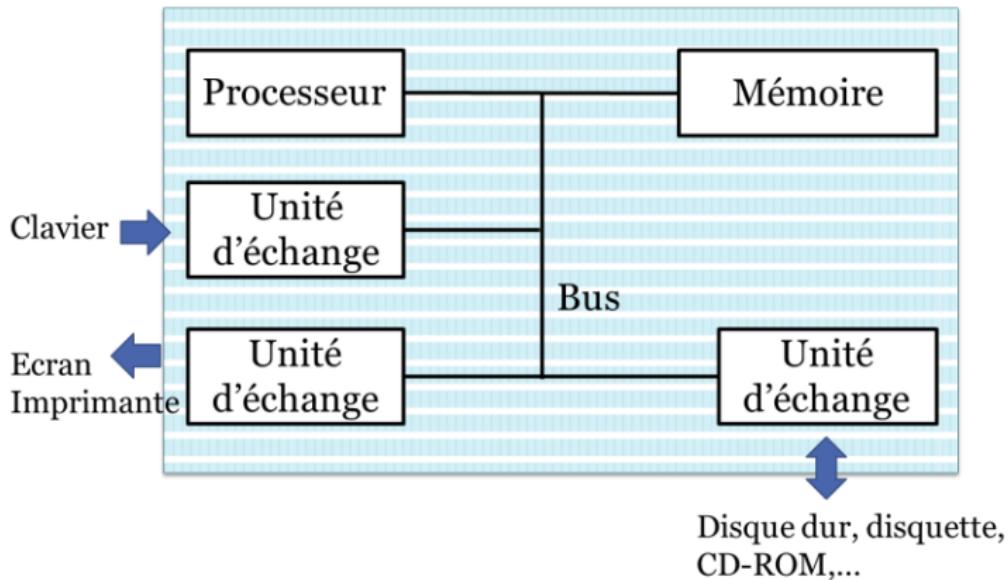
# Le "Zoo" des Systèmes d'Exploitation

- ❑ **Mainframe** : Data centers de haute performance. Capacité E/S très élevée. *UNIX/Linux*.
- ❑ **Serveur** : Services à plusieurs utilisateurs. Partage des ressources matériels et logiciels. *Linux, Windows Server, Solaris, FreeBSD*.
- ❑ **Multiprocesseur** : Plusieurs CPUs connectées. Ordinateurs personnels deviennent multiprocesseurs. *Linux, Windows*.
- ❑ **SE d'ordinateur personnel** : Fournit un bon support à un seul utilisateur. *Linux, Windows, Apple OS X*.
- ❑ **Mobile** : Téléphones, appareils photo numériques. *Android, Apple iOS*.
- ❑ **Embarqué** : TVs, DVDs, voitures, reproducteurs MP3. *Android, QNX, VxWorks*.
- ❑ **Noeud-capteur** : Réseaux des capteurs minuscules. Météo, défense, etc. SE petit et simple. *TinyOS*.
- ❑ **Temps réel** : Temps comme paramètre essentiel. Systèmes de contrôle, systèmes multimedia. *RTLinux, eCos*.

# Plan

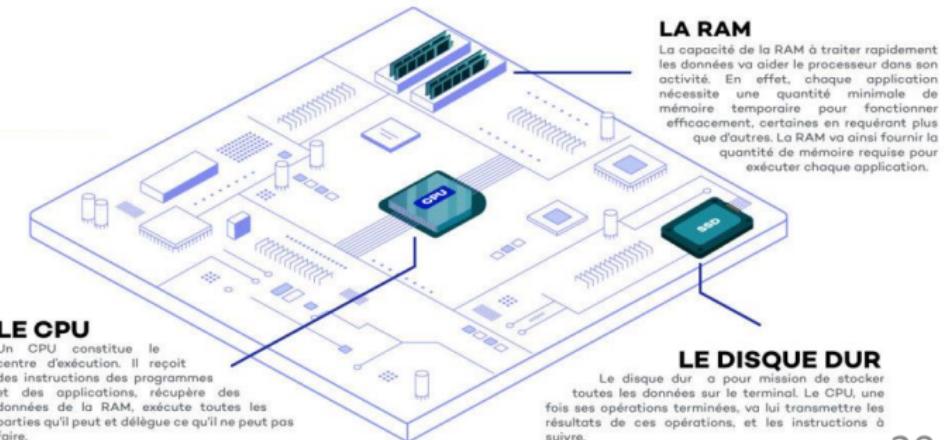
1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Révision du Matériel



# Le processeur

- ❑ **CPU (Central Processing Unit)** : Le “cerveau” de l’ordinateur.
- ❑ Exécute des instructions en **langage assembleur** des programmes **placés en mémoire centrale**.
- ❑ Répertoire fixé d’instructions, normalement incompatible avec celui d’autre famille de processeurs (Ex. Intel x86 vs. ARM).
- ❑ Registres pour stocker des variables ou des résultats temporaires (accès à mémoire très long).

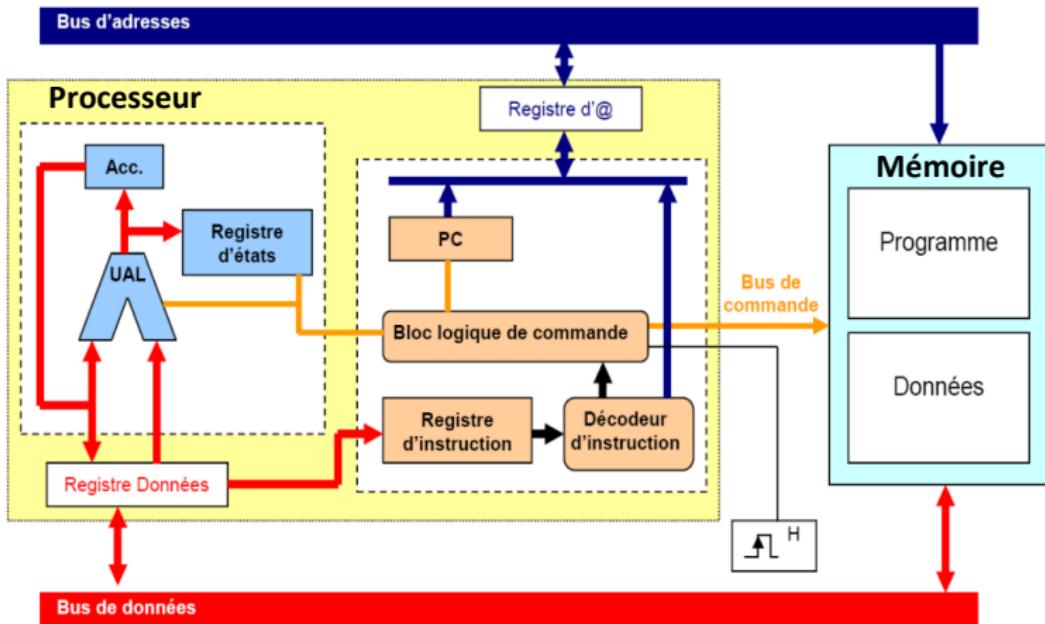


Source : <https://www.napsis.fr/cloud-lexique/cpu-et-vcpu/>

# Le processeur

- ❑ Quelques registres sont accessibles aux programmeurs :
  - **Compteur ordinal (PC)** : contient l'adresse en mémoire de la prochaine instruction à charger.
  - **Pointeur de pile** : contient l'adresse supérieure de la pile de mémoire (à revoir plus tard).
  - **PSW (Program Status Word)** : contient des informations-clés sur le fonctionnement du processeur
    - \* Valeur du compteur ordinal
    - \* Informations sur les interruptions (masquées ou non)
    - \* Mode du processeur (user ou kernel)
    - \* Priorités
    - \* Etc... (format spécifique à un processeur)

# Le processeur

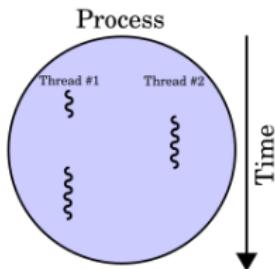


Architecture du processeur Intel x86

# Le processeur

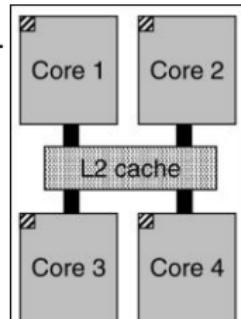
## □ Systèmes multi-thread

- Programmes avec plusieurs fils/threads d'exécution (processus "légers") qui partagent les ressources **d'un unique Coeur**.
- Multiplexage temporel des threads.
- Le SE verra plusieurs CPUs.

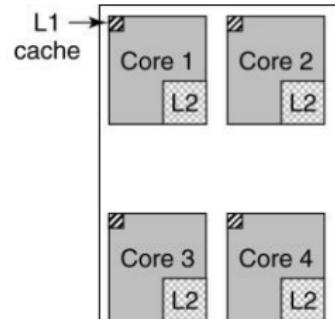


## □ Systèmes multi-noyau (multi-core) :

- **Plusieurs coeurs physiques**.
- Quad-core avec partage de mémoire cache L2 (a) vs. L2 intégrée (b).
- Intel Xeon Phi 60 *cores* !!
- GPUs (*Graphics Processing Units*).



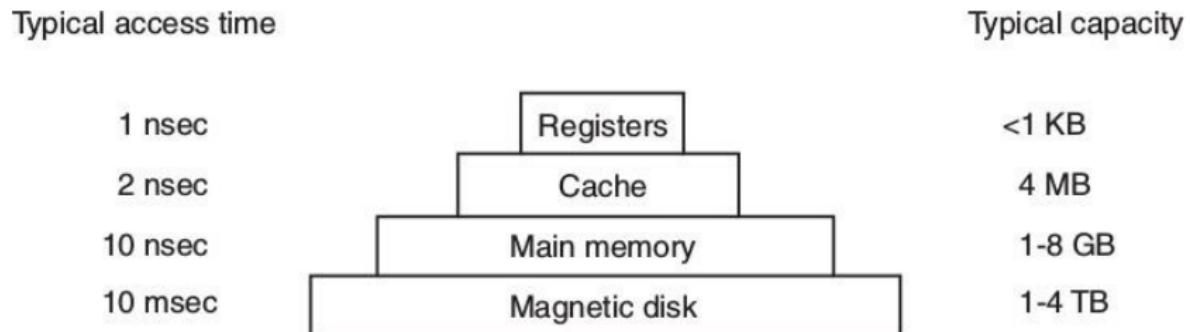
(a)



(b)

# La Mémoire

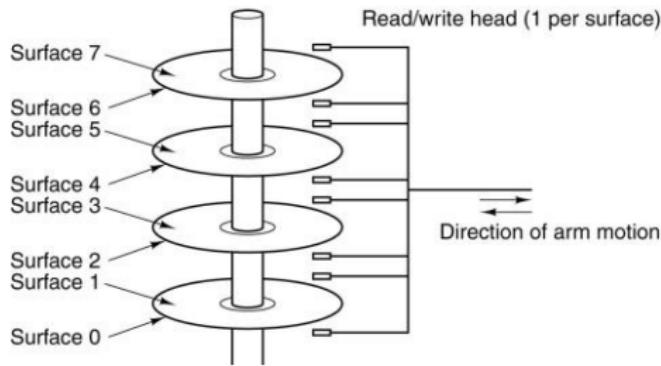
- La mémoire centrale contient les instructions et données **des applications ou des programmes** à exécuter.
- Avec le disque, la mémoire cache, elle constitue un système de hiérarchie de mémoire qui permet de rapprocher la vitesse de la mémoire centrale de celle du processeur.
- Conditions requises : rapide, grande et pas chère.
  - Approche : hiérarchie des couches



# La Mémoire

## □ Mémoire secondaire (disques durs)

- **Disque magnétique** : dispositif mécanique, avec des plateaux tournants (5400, 7200, 10.800 RPMs)
- **Dispositifs d'état solide (SSDs, Solid State Disks)** : Sans pièces mobiles. Données stockées en mémoire flash.

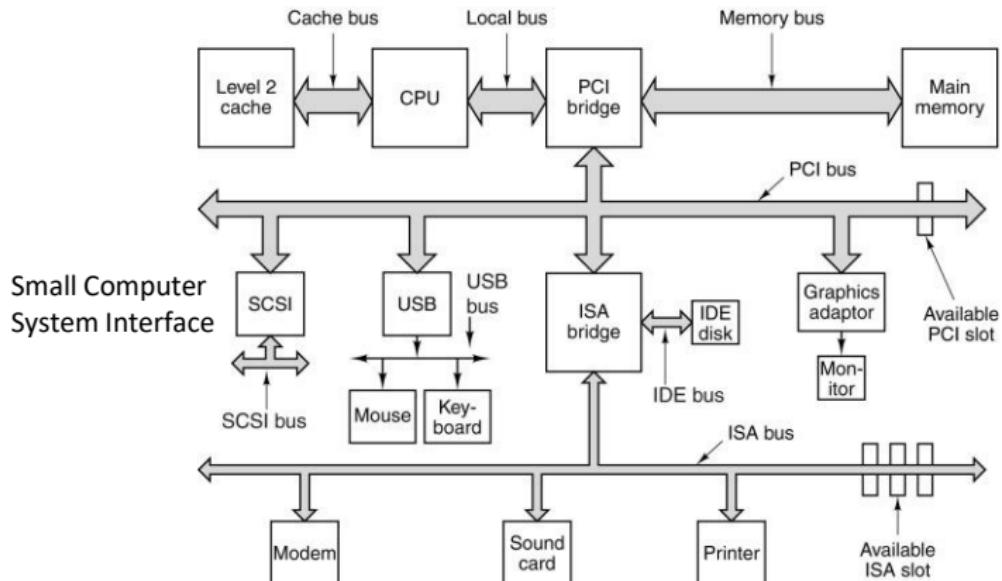


# Dispositifs d'Entrée/Sortie

- ❑ Deux parties :
  - Contrôleur : chip qui contrôle le dispositif et accepte les commandes du système d'exploitation.
  - Dispositif d'E/S.
- ❑ Chaque contrôleur est différent. Un logiciel différent pour chacun : **pilote (driver)** du dispositif.
  - Exécutés en mode kernel.
  - Peuvent être chargés dynamiquement (Linux) ou après le démarrage (MS Windows).
- ❑ Trois manières de faire E/S (exemple lecture données) :
  - Le SE appelle le *driver*, qui requiert de façon continue le dispositif pour vérifier s'il y a des nouvelles données (**polling**).
  - Le contrôleur génère une **interruption** lorsque il y a des nouvelles données à lire.
  - En utilisant un matériel spécifique : le **DMA (Direct Memory Access)**.

# Buses

- Dispositif de communication de données partagés entre les différents composants d'un système numérique.



# Plan

1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Éléments de base d'un SE

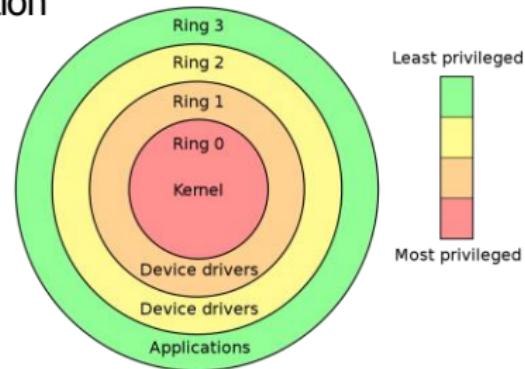
# Éléments de base d'un SE

- ❑ Processus
- ❑ Système de Fichiers
- ❑ Entrée/Sortie
- ❑ Mémoire
- ❑ Protection
- ❑ L'interprète des commandes

# Processus

## ❑ Processus : Programme qui s'exécute

- Toute l'information nécessaire pour l'exécution
- Programme, données, pile
- Compteur ordinal
- Pointeur de pile d'appel de fonction
- État des registres
- Etc.



Anneaux de priviléges pour les processeurs Intel x86 disponibles en mode protégé (source: Wikipedia).

## ❑ Multiprogrammation : Exécution de plusieurs processus en même temps.

- Arrêt/reprise des processus : sauvegarde de toute l'information sur le processus.
- Table des processus.
- Plusieurs priorités.

# Processus vs code

## ATTENTION

Ne pas confondre processus (**aspect dynamique**) avec code source d'un programme (**aspect statique**).

# Plages d'adressage

## □ Un programme s'exécute en mémoire principale.

- Plusieurs programmes en mémoire en même temps.
- Interférences à éviter en utilisant des mécanismes de protection.

## □ Mémoire virtuelle

- Adresses mémoire d'un processus distribuées en mémoire et en disque.
- Découpe la plage d'adressage du processus de la mémoire physique.
- Gérée par le SE.
- À voir plus tard...

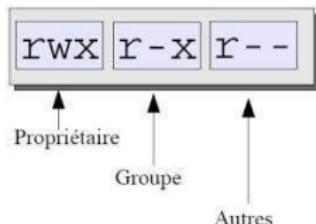
# Protection

□ Le SE gère la sécurité du système

□ Exemple : fichiers

- Code de protection de 9 bits : 3 champs de 3 bits (rwx)
- Utilisateur (propriétaire), groupe, autres
- Exemple de code de protection : rwx r-x --x
- Représentation octale :

$$\text{rwx r-x - -x} \Rightarrow 111\ 101\ 001_{bin} \Rightarrow 751_{octal}$$

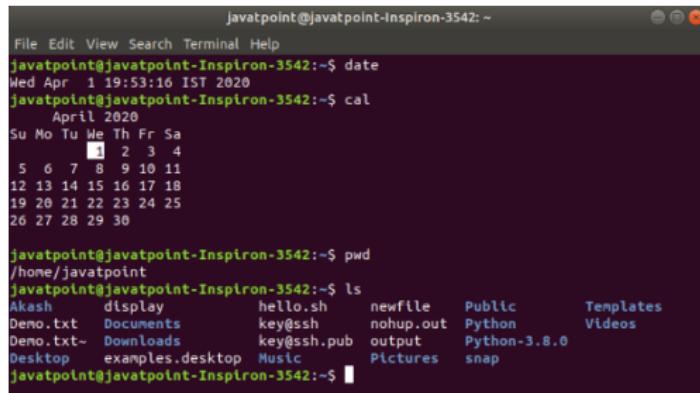


## Permission Attributes

	Files	Directories
r	Open / Read	List Contents (ls)
w	Edit	Create New, Delete, Rename
x	Execute	Access (cd)

# L'interprète des commandes (Shell)

- ❑ Ne fait pas partie du SE.
- ❑ Plusieurs shells existent : sh, csh, bash, zsh, ksh... (selon le SE).
- ❑ Exécution des commandes pour le SE.
- ❑ C'est un exemple d'utilisation des appels système.



The screenshot shows a terminal window titled "javatpoint@javatpoint-Inspiron-3542: ~". The user runs several commands:

- `date`: Returns "Wed Apr 1 19:53:16 IST 2020".
- `cal`: Prints the calendar for April 2020, showing days from 1 to 30.
- `pwd`: Returns the current working directory: "/home/javatpoint".
- `ls`: Lists files and directories in the current directory, including "Akash", "display", "hello.sh", "newfile", "Public", "Templates", "Videos", "Demo.txt", "Documents", "key@ssh", "nohup.out", "Python", "Demo.txt~", "Downloads", "key@ssh.pub", "output", "Python-3.8.0", "Desktop", "examples.desktop", "Music", "Pictures", and "snap".

# Plan

1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE

- Processus et threads
- Système de Fichiers
- Entrée/Sortie
- Plages d'adressage
- Protection
- L'interprète des commandes

## 5. Appels Système

- Interruptions

## 6. Structure des SE

## 7. Programmation de processus

# Appels Système

# Appels système

Ensemble d'instructions étendues, spécifiques d'un SE, qui constitue l'interface entre un SE et les programmes utilisateurs.

- ❑ Rappel : SE fournit abstractions aux programmes d'utilisateur et gère les ressources matérielles.
  - ▶ Exemple : Lecture d'un fichier.  
`count = read(fd, buffer, nbytes);`
- ❑ Similaire à un appel à une procédure, mais l'appel système **entre dans le kernel**.
  - Librairies avec plusieurs fonctions
- ❑ Mécanisme transparent à l'utilisateur.

# Appels système

Ensemble d'instructions étendues, spécifiques d'un SE, qui constitue l'interface entre un SE et les programmes utilisateurs.

## □ Exemples d'actions possibles

- Création d'un processus (fils) par un processus actif : `fork`
- Attendre la fin d'un processus fils : `wait`
- Destruction d'un processus : `kill`
- Mise en attente, réveil d'un processus : `sleep`, `wait`
- Suspension et reprise d'un processus grâce à l'ordonnanceur de processus (`scheduler`)
- Demande de mémoire supplémentaire ou restitution de mémoire inutilisée : `allocation dynamique`, `malloc`, `free`.
- etc.

## Interruptions

Puisque le processeur est en permanence prêt à exécuter des instructions...

Comment peut-il prendre en compte les évènements extérieurs ?

- **Évènement extérieurs** : Requête d'un périphérique, appui sur un bouton poussoir, passage d'un objet devant un capteur, ...
  
- À chacun de ces événements correspond une tâche à exécuter par le processeur. Cette tâche est codée sous forme d'une procédure (**appel système**).

## Interruptions

Puisque le processeur est en permanence prêt à exécuter des instructions...

Comment peut-il prendre en compte les évènements extérieurs ?

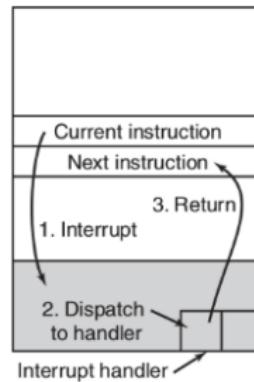
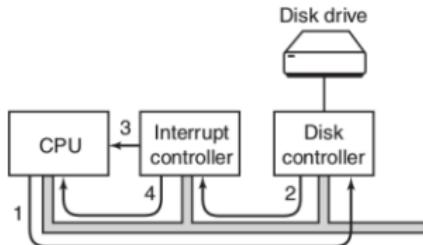
- Pour pouvoir exécuter cette procédure il faut que se produise une **rupture de séquence**. Cette rupture doit avoir lieu dans un délai assez court.
- Le processeur dispose d'une entrée spéciale, appelée **IRQ** (*Interrupt ReQuest*), associée à un bit appelé **bit d'interruption**. Avant de passer à l'instruction suivante, le processeur teste l'état de ce bit. S'il est à **1** le processeur est informé d'une **demande d'interruption**.

À voir plus tard ... AdO

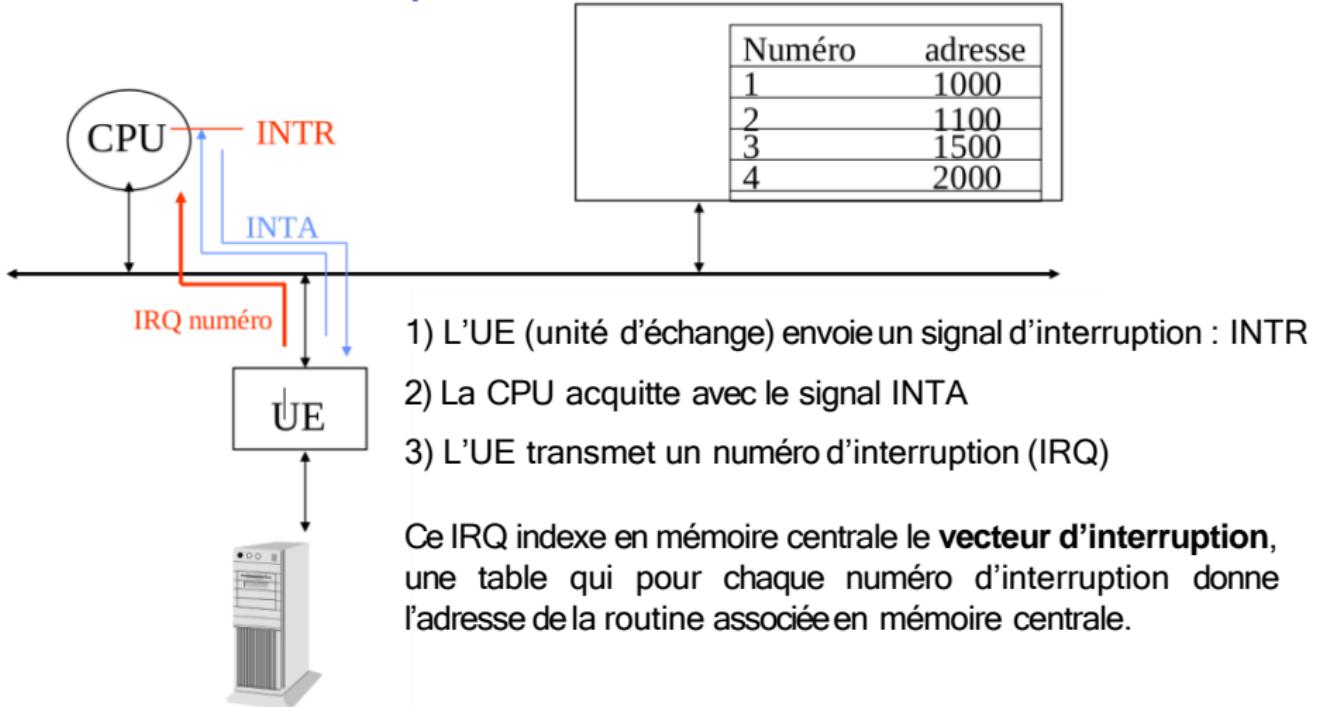
# Gestion des interruptions

□ Déroulement :

- 1) Arrêt du processus en cours à la fin de l'instruction courante.
  - 2) Sauvegarde de l'état présent du processeur : le contexte (copie de l'état des registres hardware au PCB (Bloc de Contrôle de Processus), une structure logicielle qui représente l'état courant du processus).
  - 3) Exécution d'un sous programme dépendant de la nature de l'interruption.
  - 4) Restauration du contexte ou état du processeur.
- Interruption (1) ⇒ exécution d'un sous programme (*interrupt handler* ou gestionnaire de l'interruption (2) )



# Gestion des interruptions



INTA - Signal d'accusé de réception d'interruption. RESET IN est utilisé pour réinitialiser le microprocesseur en mettant le compteur de programme à zéro. RESET OUT est utilisé pour réinitialiser tous les périphériques connectés lorsque le microprocesseur est réinitialisé.

# Classification des Interruptions

## □ Interruptions externes :

- **Matérielles** : panne, intervention de l'opérateur, dues aux périphériques ou à des dispositifs extérieurs, ...
- **Logicielles** : déclencher une interruption à l'aide d'une instruction spéciale.

## □ Interruptions internes :

- Déroulements qui proviennent d'une situation exceptionnelle ou d'une erreur liée à l'instruction en cours d'exécution (division par 0, débordement, dépassement de capacité, erreur d'adressage, ...)
- Ces déroutements génèrent des interruptions (TRAP) pour passer de mode utilisateur à mode kernel, normalement liées à des erreurs irrécouvrables.

## Traitement effectif des interruptions

- ❑ Affectation d'un **numéro de priorité** par niveau d'interruption.
- ❑ Permet d'ordonner les traitements lors des cascades d'interruptions.
- ❑ Nécessité de **retarder** ou **d'annuler** la prise en compte d'un signal.
- ❑ Techniques : masquage **et** désarmement
  - **Masquage de signaux**
    - \* Ignore temporairement la prise en compte des interruptions d'un niveau
    - \* Pour cela, on positionne un indicateur spécifique dans le PSW
    - \* Possibilité de masquer d'autres niveaux
    - \* Démasquage des signaux
    - \* Prise en compte des interruptions survenues pendant ce temps
  - **Désarmement de signaux**
    - \* Supprimer la prise en compte du niveau d'interruption
    - \* Réactivation : réarmement

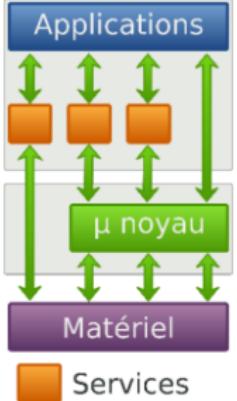
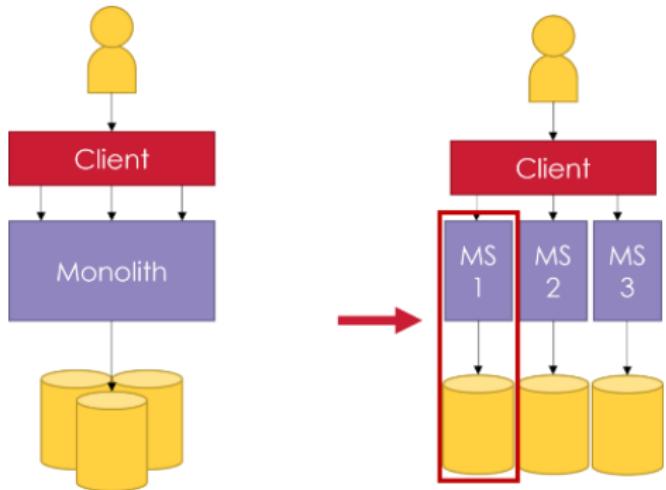
# Plan

1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Structure des SE

# Catégories de SE

- Systèmes monolithiques
- Micro-noyau
- Autres



Source : Wikipédia

## Systèmes monolithiques

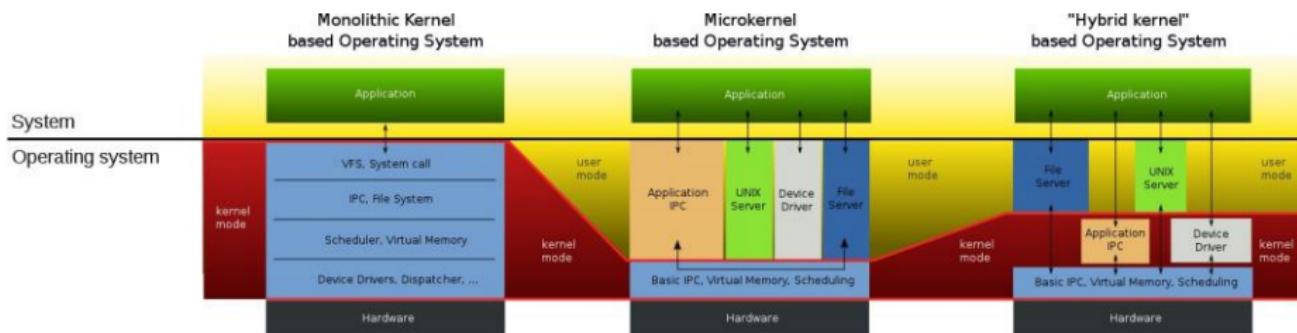
- ❑ Tout le SE s'exécute comme **un seul programme** en mode kernel.
- ❑ Le SE est un **ensemble de procédures** compilés dans un seul programme exécutable (et assez grand).
- ❑ Chaque procédure dans le système peut appeler les autres sans restrictions. Inconvénients :
  - Système peu maniable, difficile à comprendre.
  - Si une procédure s'écrase, cela écrasera tout le SE.
- ❑ Structure la plus commune.

## Systèmes monolithiques

- Démarche d'un appel à procédure dans un système monolithique :
  - 1) Appels systèmes : services offerts par le SE aux applications utilisateurs.
  - 2) Les paramètres de l'appel sont récupérés de la pile (*stack* en anglais).
  - 3) Dépôts dans un registre les paramètres.
  - 4) Passage en mode kernel (superviseur).
  - 5) SE analyse et exécute la procédure équivalente à l'appel système.
  - 6) Commutation dans le mode utilisateur.
- Peut inclure facilement des services spécifiques :
  - Chargement de module dynamique (Par exemple, pilotes).
  - En Unix : librairies partagées
  - En Windows : DLLs (*Dynamic-Link Libraries*)

# Micro-noyau (*Microkernels*)

- Systèmes monolithiques : volumineux (un service rare, doit être présent) ⇒ Source d'erreurs.
- *Micro-noyau* ⇒ Réduction du noyau aux procédures essentielles.
- Objectif : réduire la taille du noyau et obtenir une fiabilité plus grande.
- Services reportés dans des programmes externes (*modules*).
- Services à l'extérieur du noyau (dans l'espace utilisateur).



Source : Wikipedia

## Autres

- ❑ Systèmes **en couche**, comme le *THE OS* (*Technische Hogeschool Eindhoven / Eindhoven University of Technology*) par DIJKSTRA
- ❑ Modèle **Client-Serveur** :
  - 2 types de processus : clients et serveurs.
    - \* Serveurs : dont chacun fournit un service.
    - \* Clients : qui utilisent ces services.
    - \* Communication par passage de messages.
- ❑ **Virtual machines** : à voir plus tard dans ce module.
- ❑ **Exokernels** :
  - Chaque utilisateur reçoit un sous-ensemble des ressources (disk, mémoire, etc.)
  - L'Exokernel est un processus qui, s'exécutant en mode superviseur, gère la répartition des ressources.

# Plan

1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
  - Processus et threads
  - Système de Fichiers
  - Entrée/Sortie
  - Plages d'adressage
  - Protection
  - L'interprète des commandes
5. Appels Système
  - Interruptions
6. Structure des SE
7. Programmation de processus

# Programmation de processus

# Les processus

## ❑ Informations sur les processus UNIX / Linux

- Notion de père/fils
- `pid` : identifiant unique du processus. Chaque processus est identifié par son PID unique
- `ppid` : identification du processus père
- Propriétaire
- Priorité
- etc.

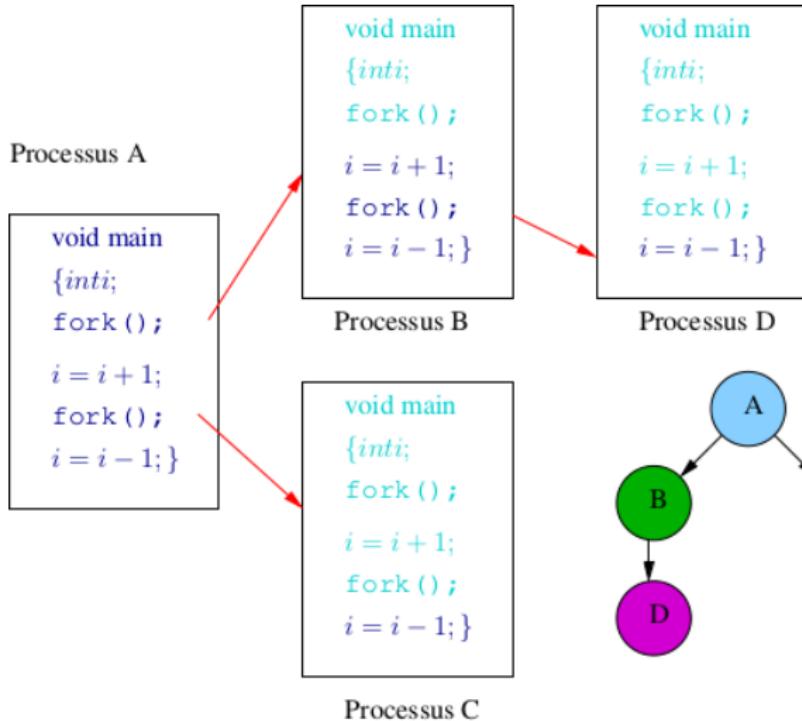
## ❑ Hiérarchie des processus

- Lorsqu'un processus crée un autre processus, **les processus père et fils continuent d'être associés** d'une certaine manière. **Chaque processus connaît le PPID de son parent.**
- Le **processus enfant** peut lui même **créer plusieurs processus**, formant une hiérarchie de processus.
- Un **processus a un seul parent** mais peut avoir aucun ou plusieurs fils.

## Création d'un processus

- ❑ Appel à l'appel système – **fonction fork()** **en langage C**. Création par clonage (mitose) : un processus (**le père**) demande, en appelant la primitive `fork()`, la création dynamique d'un nouveau processus (**le fils**). Le fils s'exécute ensuite de façon concurrente avec le père.
- ❑ `pid_t fork (void)`
  - Processus fils
    - ★ partage le segment de texte du père
    - ★ dispose d'une copie de son segment de données
    - ★ hérite du terminal de contrôle
    - ★ hérite d'une copie des *file descriptor* ouverts
    - ★ n'hérite pas du temps d'exécution, ni de la priorité
    - ★ n'hérite pas des signaux en suspend
  - Appel
    - ★ Retourne 0 au fils
    - ★ Retourne le PID du fils au père, ou -1 si échec
- ❑ Un processus avec tous ses descendants forment un groupe de processus représenté par un **arbre de processus**.

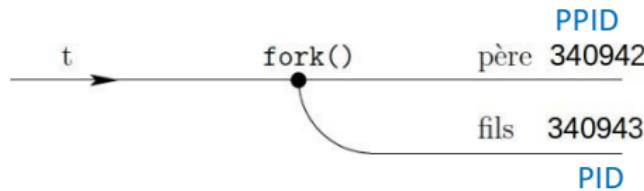
# Création d'un processus



# Exemple

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     pid_t p;
8
9     p = fork();
10
11    switch(p) {
12        case 0 :
13            printf("Je suis le fils : mon PID est %d et mon PPID est %d\n", getpid(), getppid());
14            break;
15
16        case -1 :
17            perror("Erreur de creation de processus avec fork");
18            break;
19
20        default :
21            printf("Je suis le pere : mon PID est %d et mon PPID est %d\n", getpid(), getppid());
22            break;
23    }
24    return 0;
25 }
```



Compilation : gcc -o useoffork useoffork.c

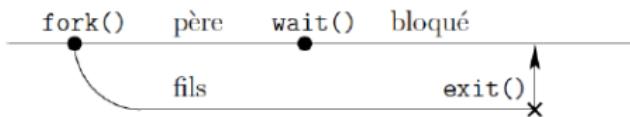
# Synchronisation

- `exit(int etat)`
  - Termine un processus normalement, `etat` est un octet (donc valeurs possibles : 0 à 255) renvoyé dans une variable du type `int` au processus père.
  - Usage de la variable `etat` : 0 → ok ; ≠ 0 → code erreur
  - Constantes `stdlib.h` : `EXIT_SUCCESS`, `EXIT_FAILURE`
- `pid_t wait(int *status)`
- `pid_t waitpid (pid_t pid, int *status, int options)`
  - ▶ Suspend le processus jusqu'à la terminaison de l'un de ses fils de fils
  - ▶ Achèvement du père : fils pris en charge par le processus `init` ([processus de PID = 1](#))
  - ▶ `status` : valeur de `exit` ou autre (dans le cas d'un signal)
  - ▶ `wait` attend la fin de n'importe quel fils et renvoie son PID ou -1 dans le cas où il n'y a pas (ou plus) de fils
  - ▶ `waitpid` pour attendre un processus particulier dont on connaît son `pid`

# Exemple : Synchronisation

```

1 #include <unistd.h>      /* Symbolic Constants */
2 #include <sys/types.h>    /* Primitive System Data Types */
3 #include <stdio.h>        /* Input/Output */
4 #include <sys/wait.h>     /* Wait for Process Termination */
5 #include <stdlib.h>       /* General Utilities */
6
7 int main()
8 {
9     int retval; int status;
10
11     pid_t p = fork();
12
13     if (p == 0) {
14         sleep(1); /* sleep for 1 second */
15         printf("CHILD: Enter an exit value (0 to 255): ");
16         scanf("%d", &retval);
17         exit(retval);
18
19     } else if (p > 0) {
20         printf("PARENT: I will wait for my child to exit.\n");
21         wait(&status);
22         printf("PARENT: Child exit code is: %d\n", WEXITSTATUS(status));
23         exit(0);
24
25     } else { exit(-1); }
26 }
```



Code équipe TEAMS :

lgjtyyj