Système d'exploitation

Introduction aux SE
Programmation de processus

Juan Angel Lorenzo del Castillo

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

ING1 Informatique – Mathématique appliquée

2023–2024

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.

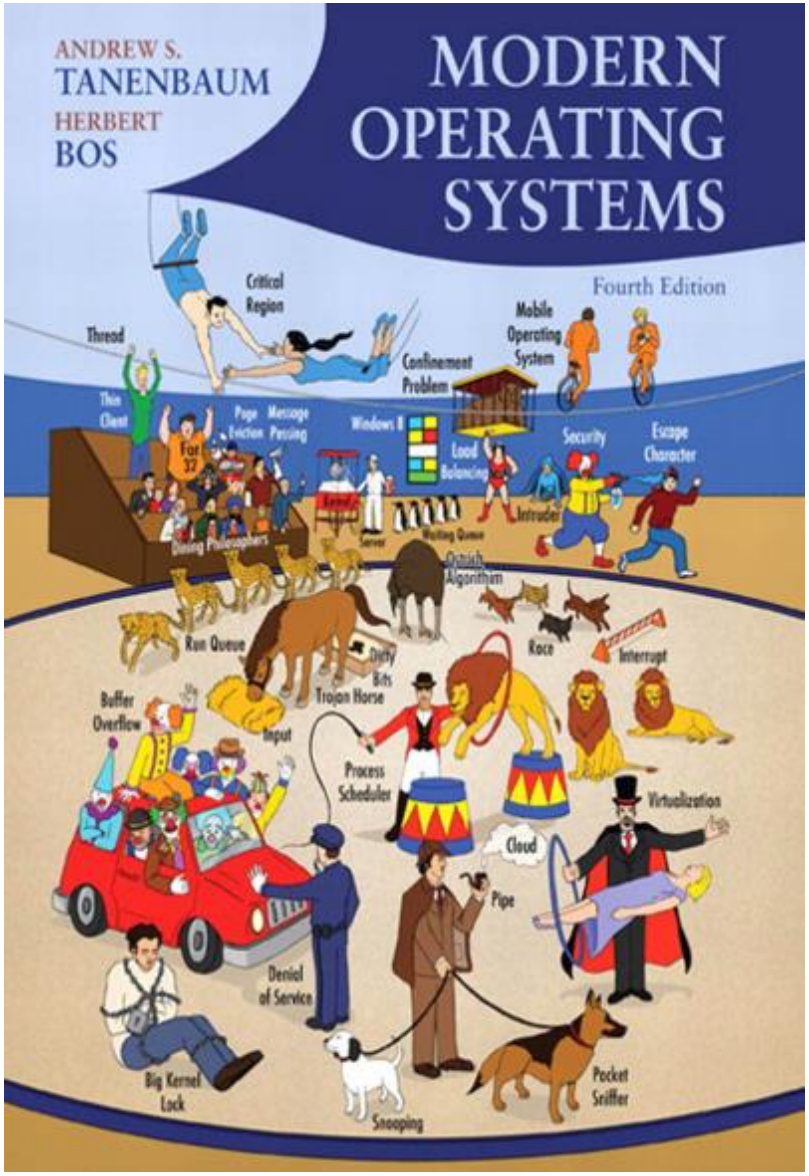
Objectifs

- 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
- Examens des années précédentes disponibles sur Teams

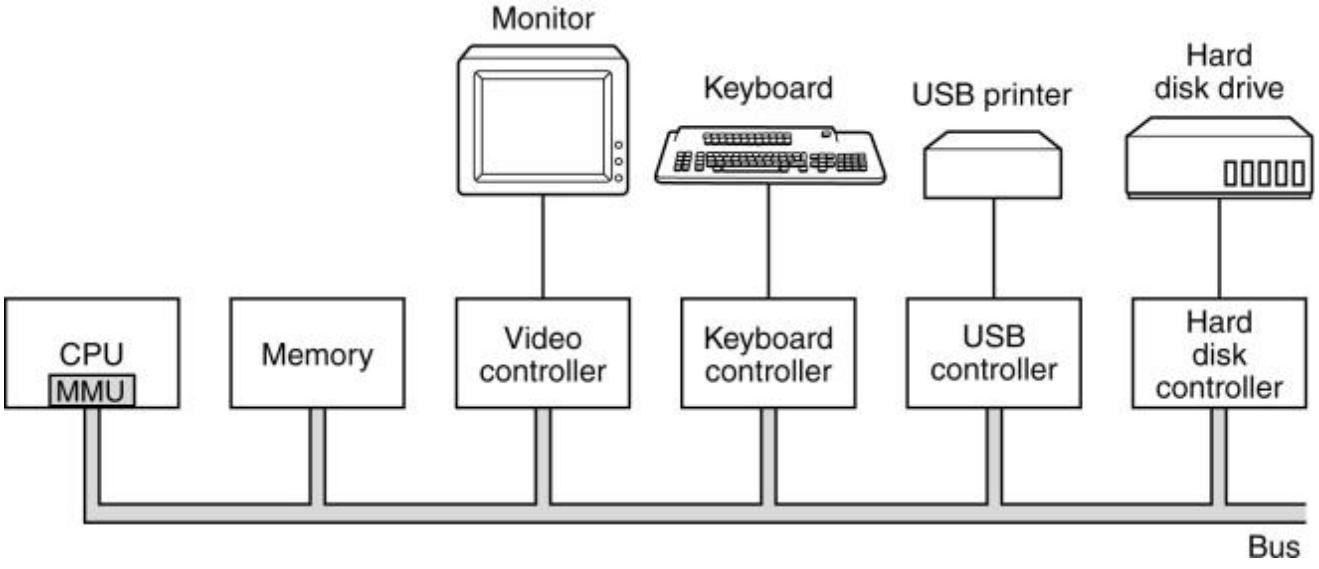
Plan

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

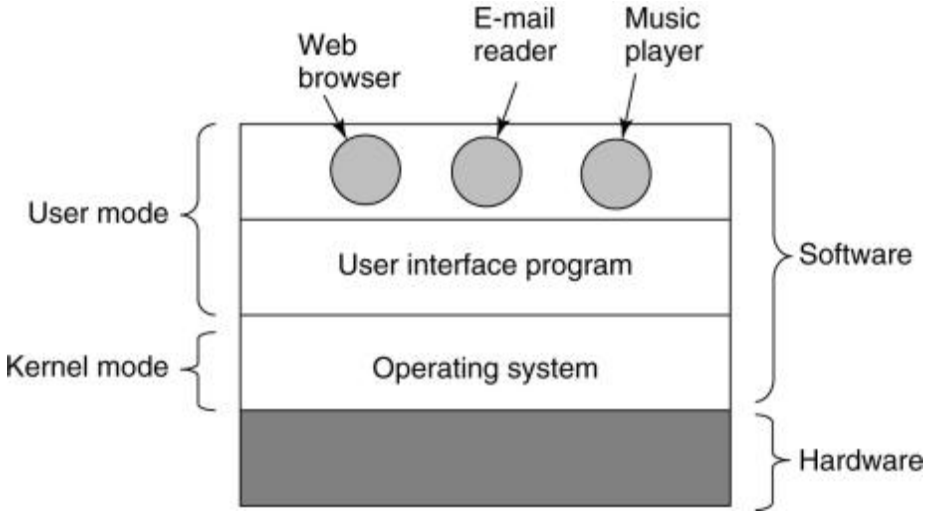
L'ordinateur moderne, un système complexe

- 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 ?
  - Ou à la mémoire (physiquement) ?
  - Comment gérer des erreurs ?
- Ces problèmes font partie du rôle d'un SE

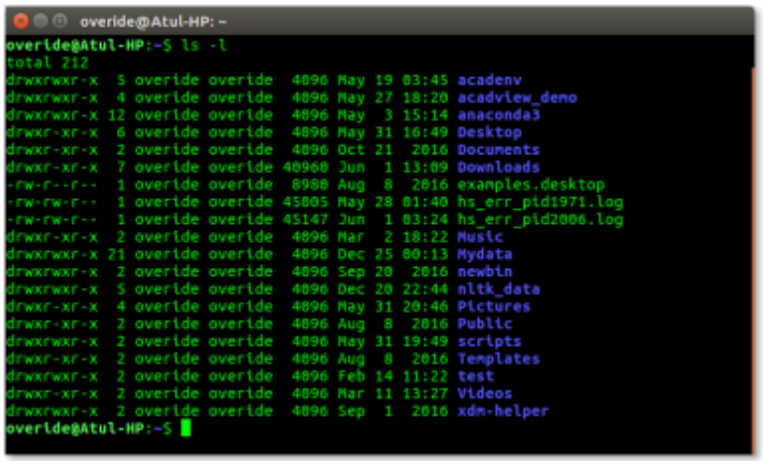
Système d'exploitation

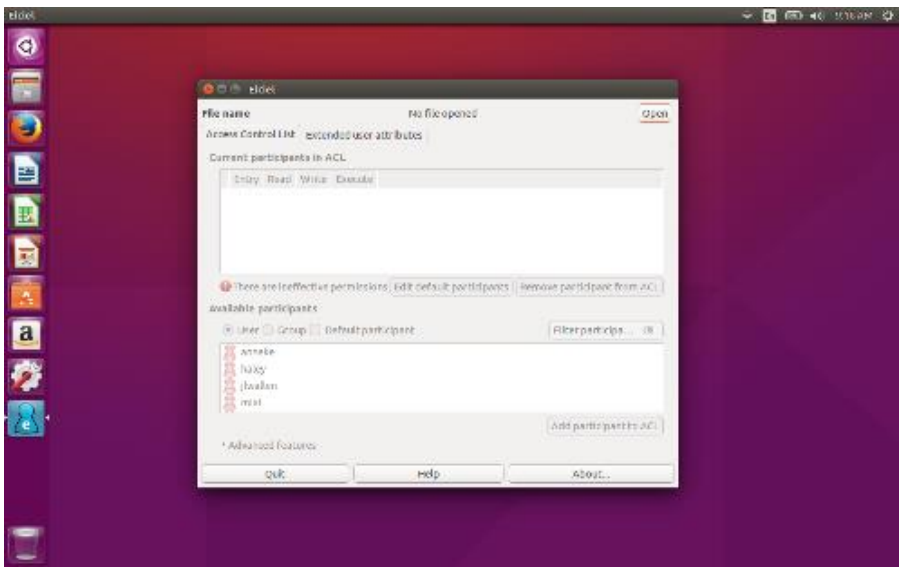
Couche logiciel qui s'occupe de :

- 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





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.*

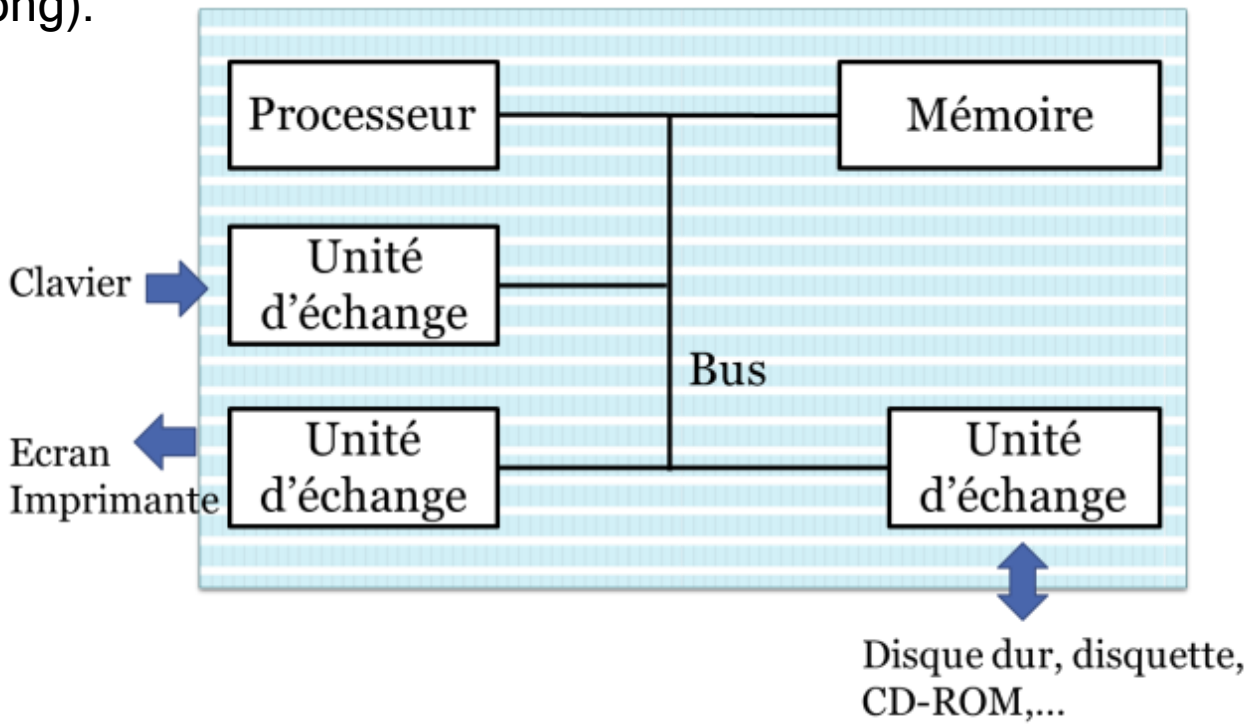
Plan

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

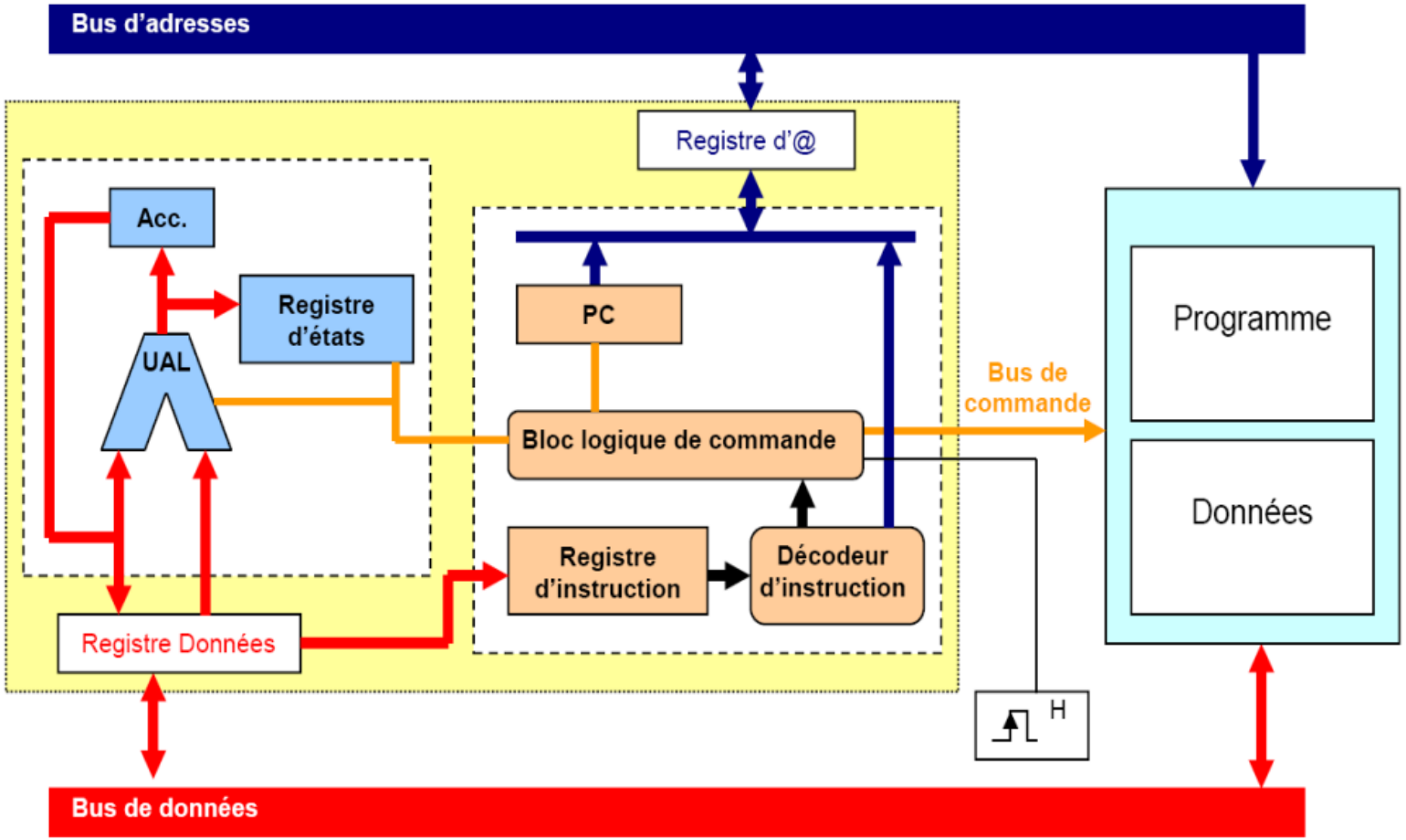
- **CPU (*Central Processing Unit*)** : Le “cerveau” de l’ordinateur.
- Exécute des instructions en langage assembleur des programmes **placés en mémoire centrale**.
- Répertoire fixé d’instructions, normalement incompatible avec celui d’autre famille de processeurs (Ex. Intel x86 vs. ARM).
- Registres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



Le processeur

- Quelques registres sont accessibles aux programmeurs :
  - **Compteur ordinal** : contient l'adresse en mémoire de la prochaine instruction à charger.
  - **Pointeur de *stack*** : 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)

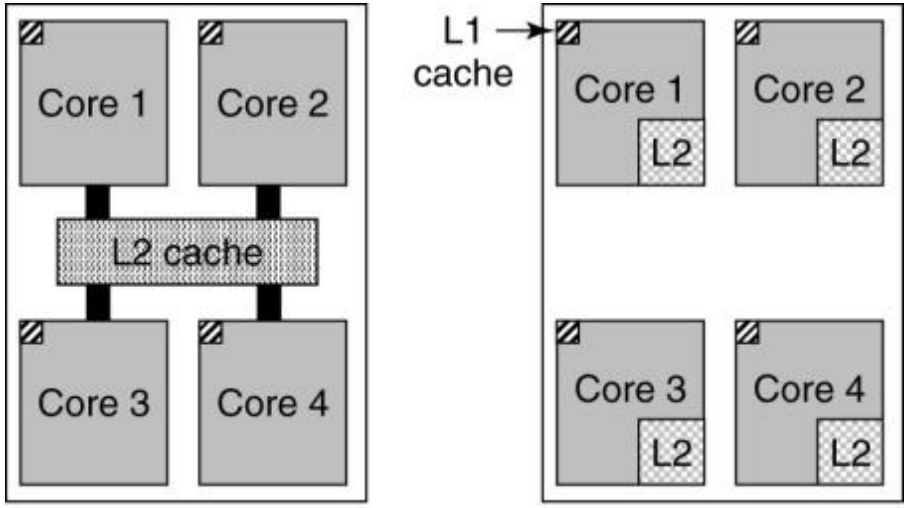
Le processeur



Architecture processeur Intel x86

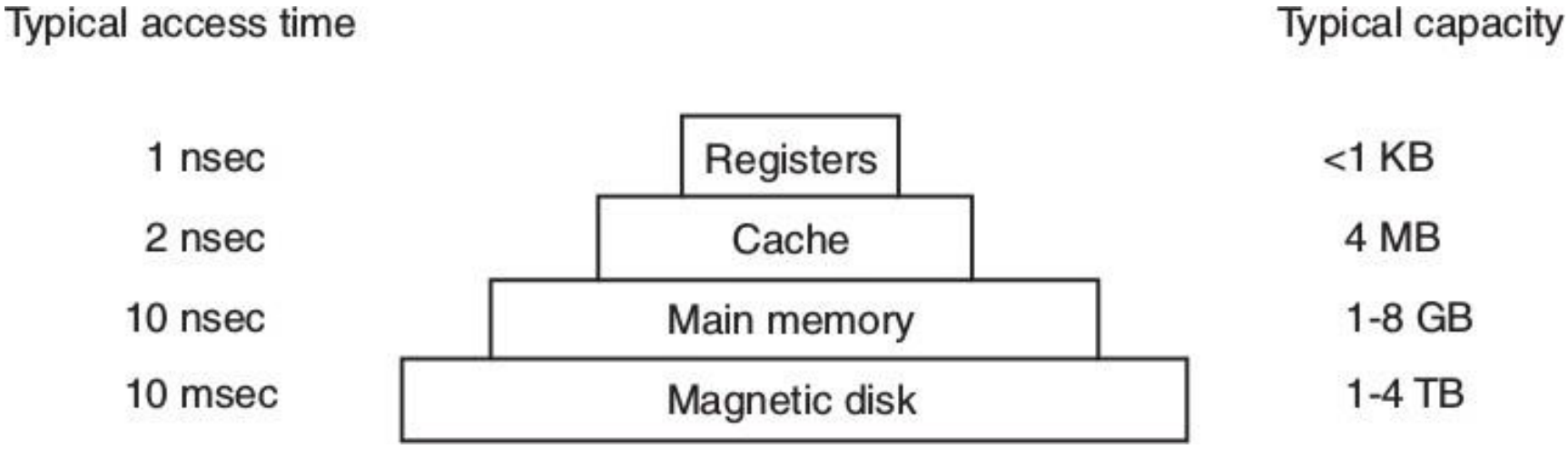
Le processeur

- Systèmes multi-thread
  - Programmes avec plusieurs fils 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*)



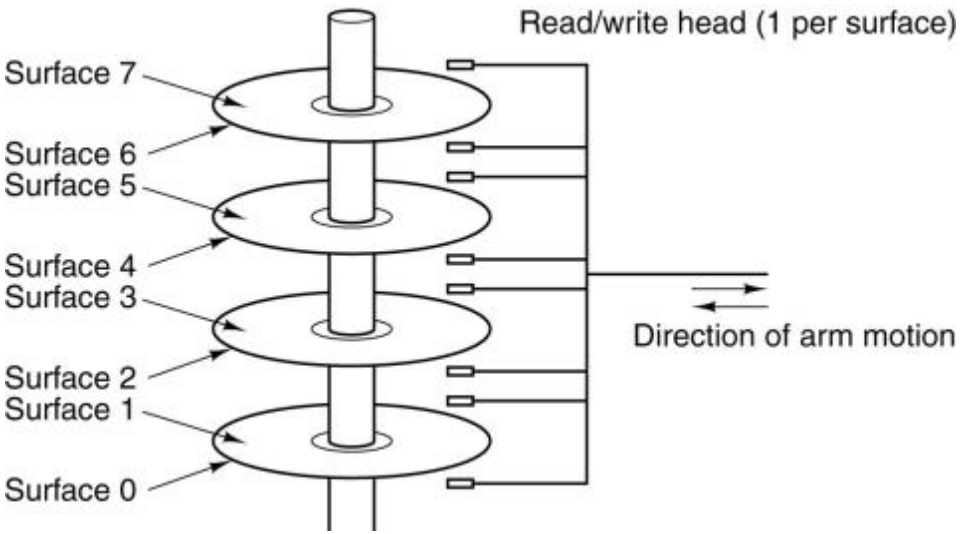
La Mémoire

- 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

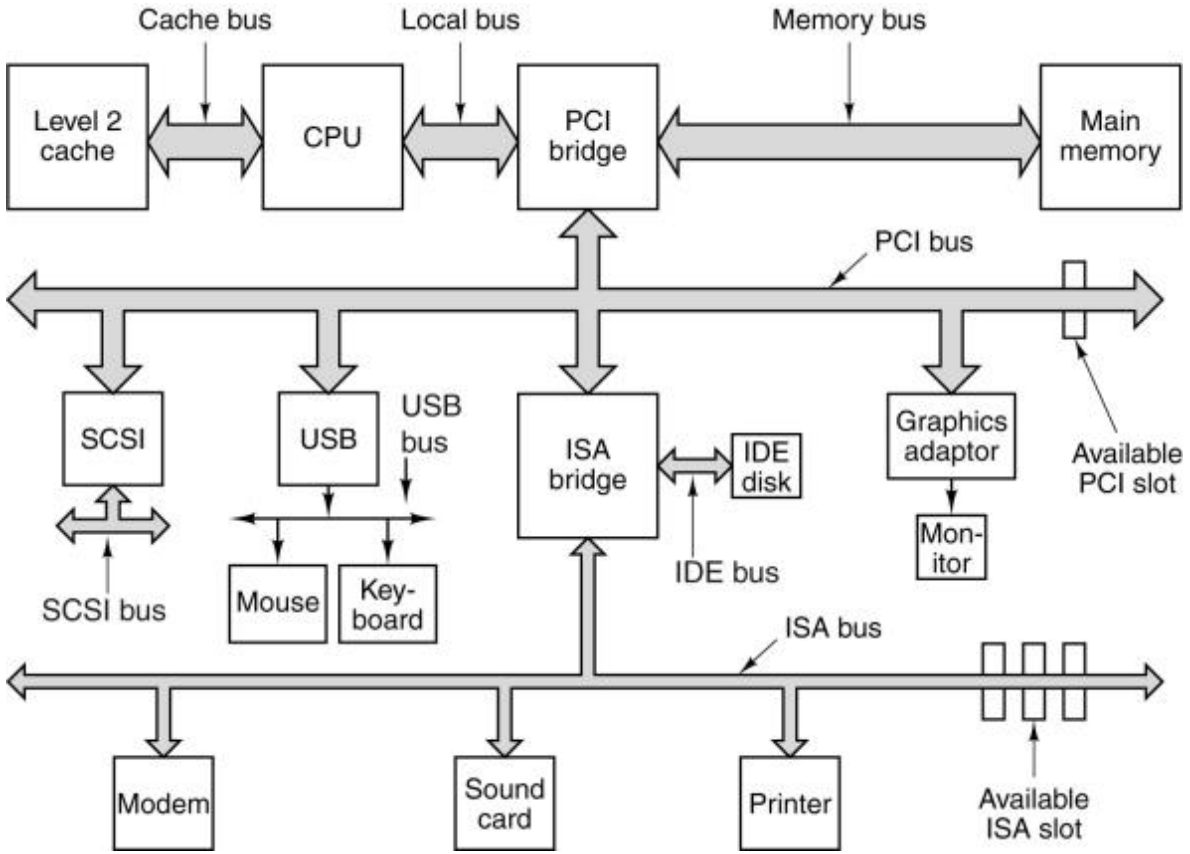
- Deux parties :
  - Contrôleur : chip qui contrôle le dispositif et accepte les commandes du système d'exploitation.
  - Dispositif d'E/S.

- Chaque contrôleur est différent → 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

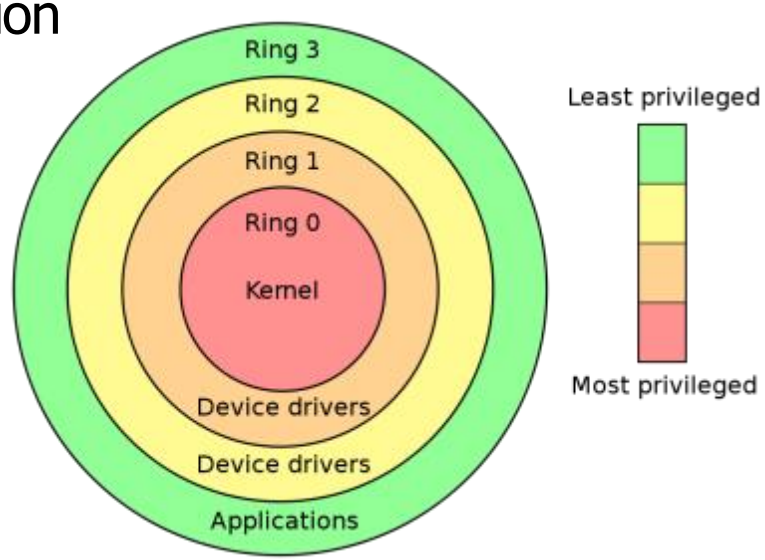
Éléments de base d'un SE

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 privilèges pour les processeurs 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

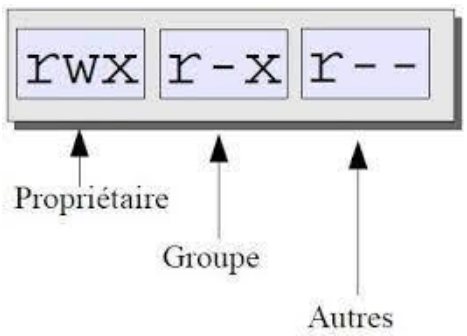
- Un programme s'exécute en mémoire principale.
  - Plusieurs programmes en mémoire en même temps.
  - Interférences à éviter en utilisant des mécanismes de protection.
- **Mémoire virtuelle**
  - Adresses mémoire d'un processus distribuées en mémoire et en disque.
  - Découple la plage d'adressage du processus de la mémoire physique.
  - Gérée par le SE.
  - À voir plus tard...

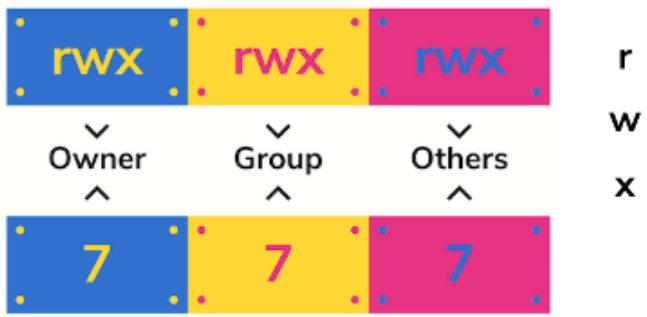
Protection

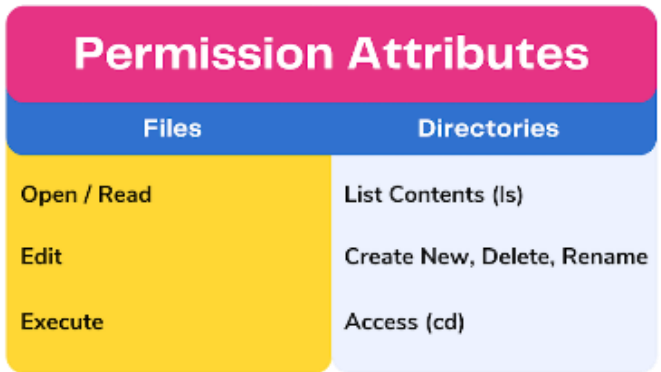
- Le SE gère la sécurité du système

- 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 :

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

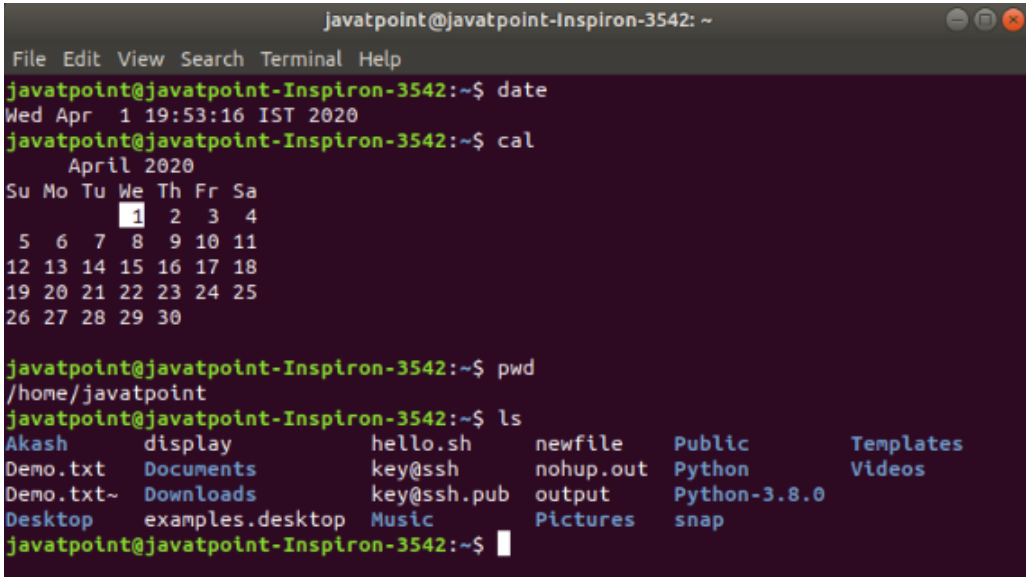






L'interprète des commandes (Shell)

- 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



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.

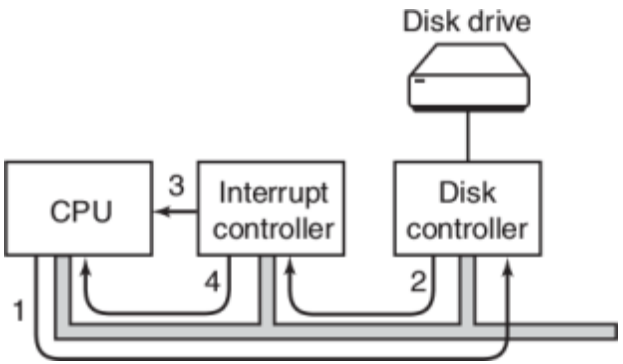
- Le processeur dispose d'une entrée spéciale, appelée **IRQ** (***I****nterrupt* ***R****e****Q****uest*), associée à un bit appelé **bit d'interruption**. Avant de passer à l'instruction suivante, le processeur teste l'état de ce bit. S'il est à 1 le processeur est informé d'une demande d'interruption.

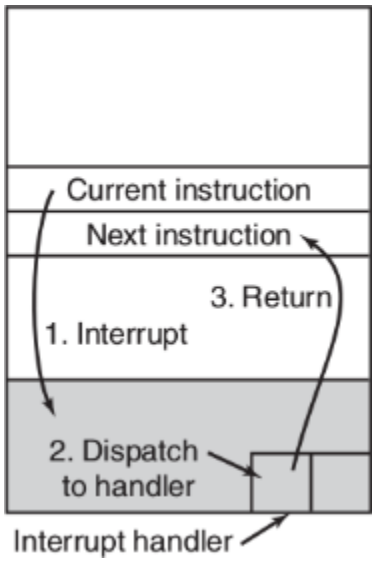
À voir plus tard ... AdO

Gestion des interruptions

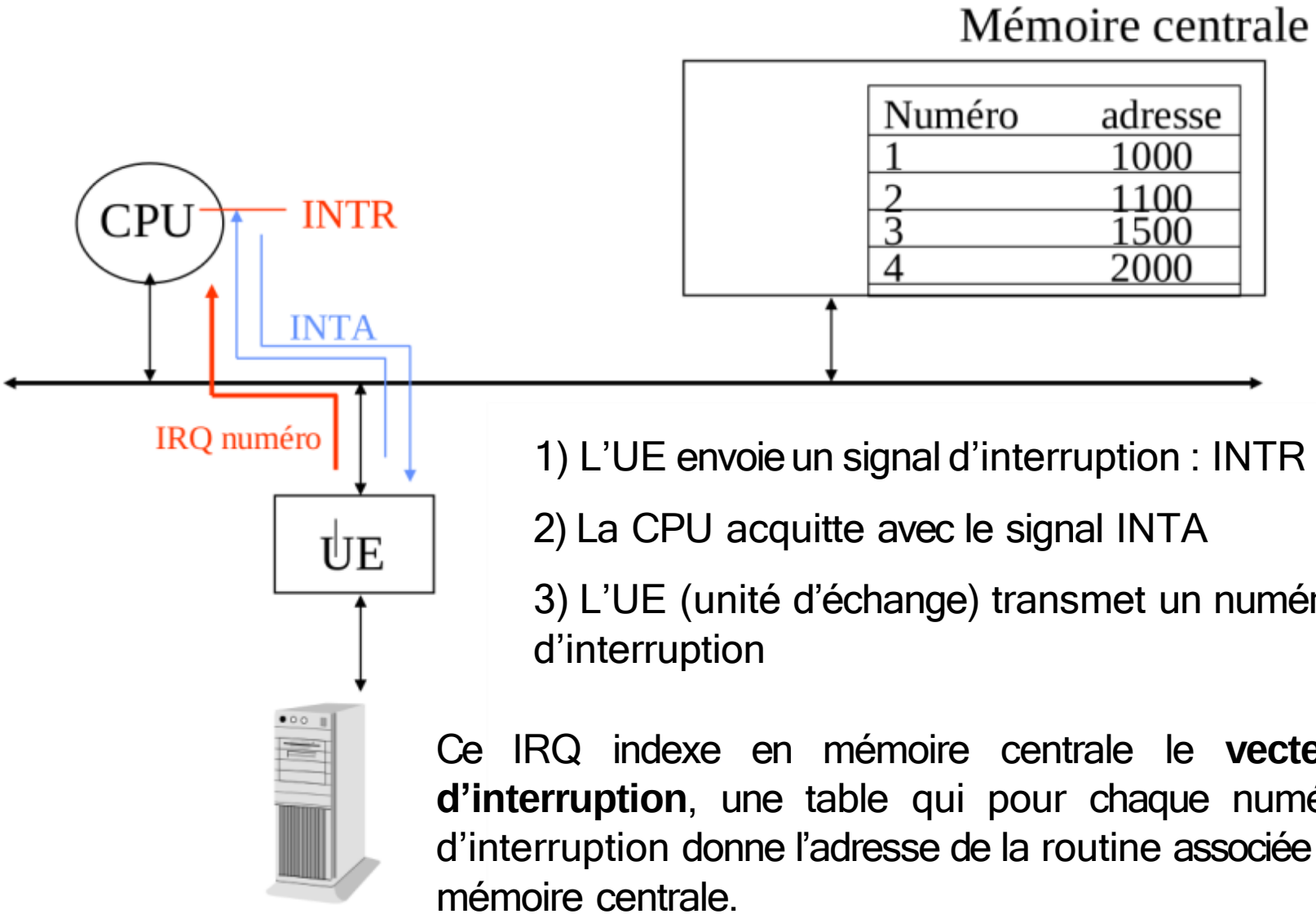
- Déroulement :
  1) Arrêt du processus en cours à la fin de l'instruction courante.
  2) Sauvegarde de l'état présent du processeur : le contexte (copie de l'état des registres hardware au PCB (Bloc de Bontrôle de Processus), une structure logicielle qui représente l'état courant du processus).
  3) 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 (*interruption handler* ou gestionnaire de l'interruption (2))





Gestion des interruptions



1) L'UE envoie un signal d'interruption : INTR

2) La CPU acquitte avec le signal INTA

3) L'UE (unité d'échange) transmet un numéro d'interruption

Ce IRQ indexe en mémoire centrale le **vecteur d'interruption**, une table qui pour chaque numéro d'interruption donne l'adresse de la routine associée en mémoire centrale.

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 :**
  - **Déroutements** 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
    - Reactivation : réarmement

Exemple : Interruption horloge

Mode utilisateur

protection

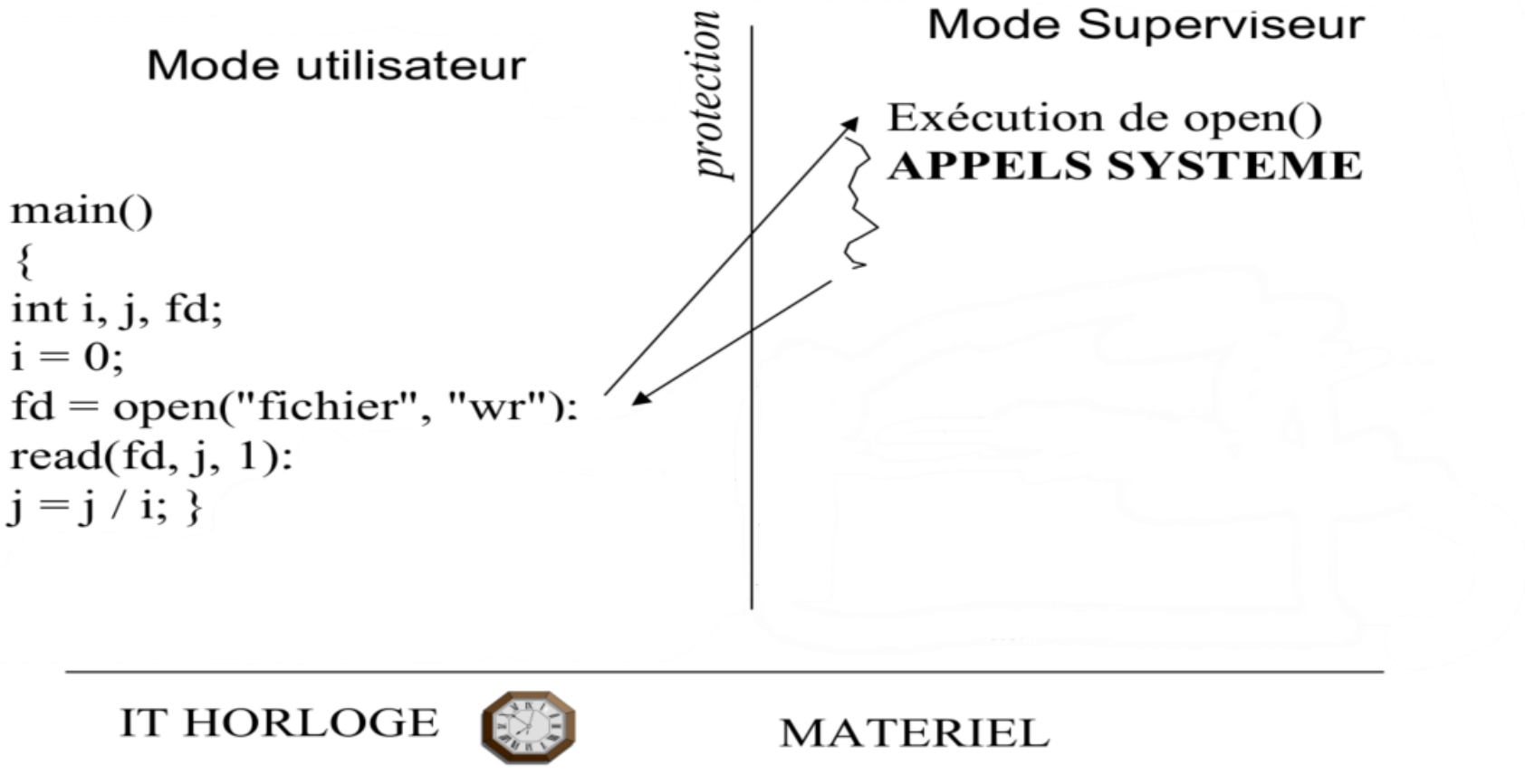
Mode Superviseur

```
main()
{
int i, j, fd;
i = 0;
fd = open("fichier", "wr");
read(fd, j, 1);
j = j / i; }
```

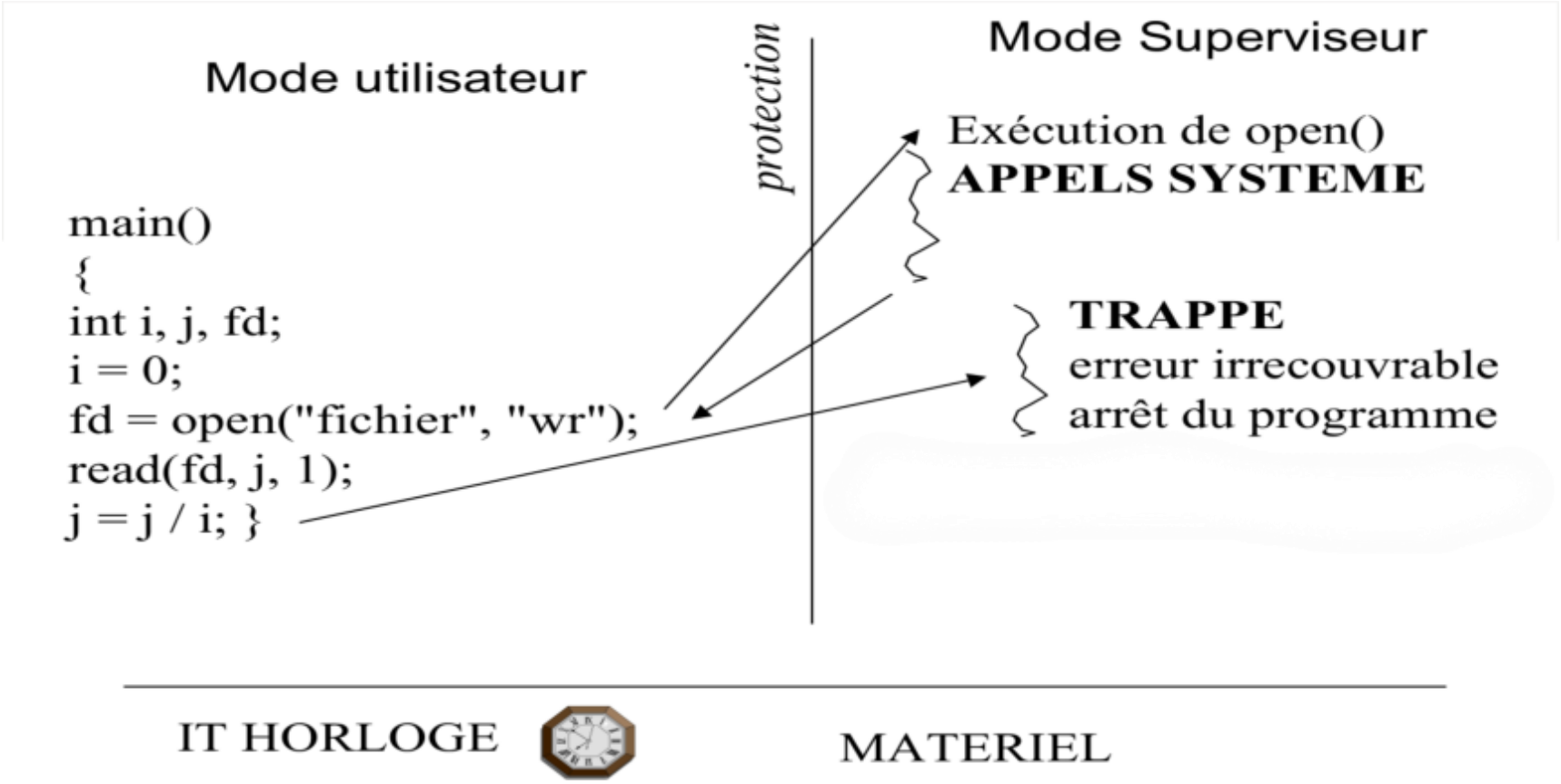
IT HORLOGE

MATERIEL

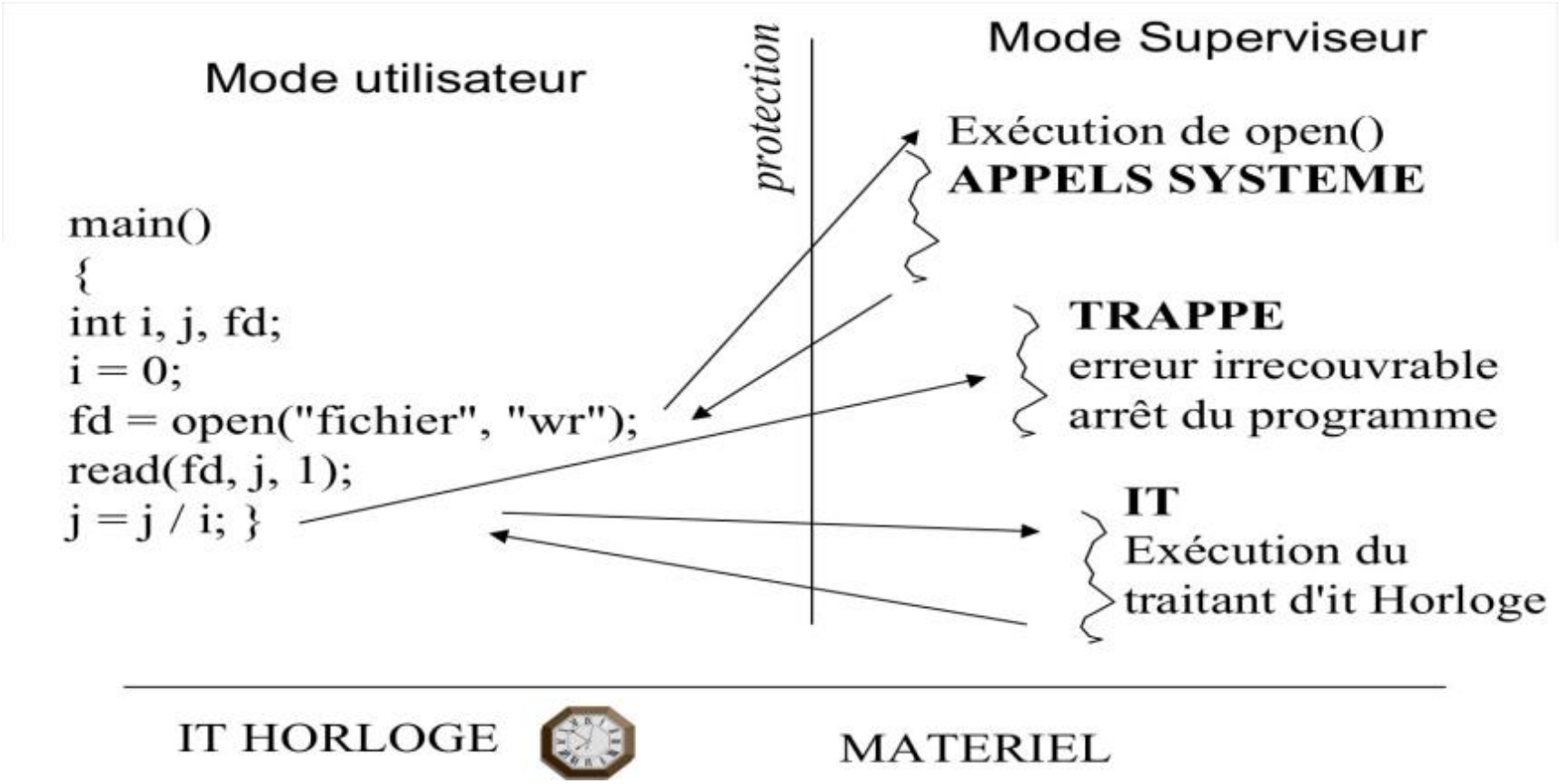
Exemple : Interruption horloge



Exemple : Interruption horloge



Exemple : Interruption horloge



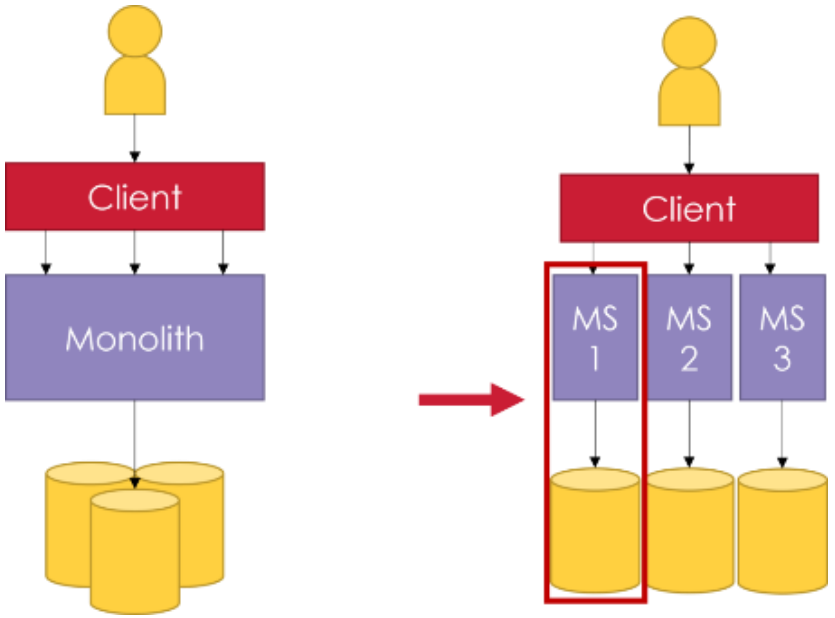
Plan

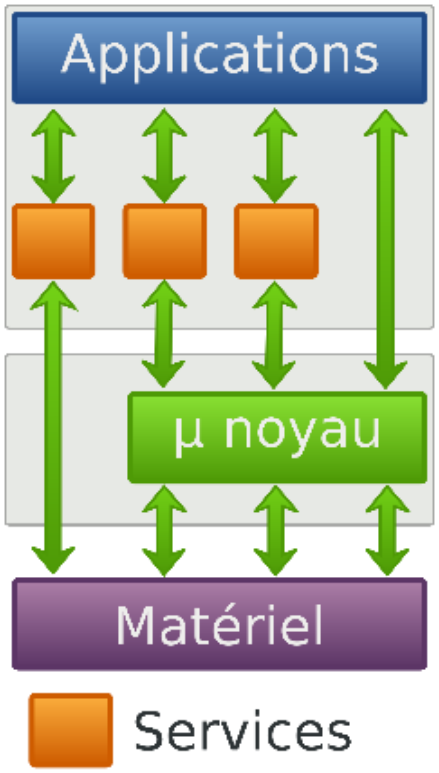
1. Introduction
2. Le Système d'Exploitation
3. Révision du Matériel
4. Éléments de base d'un SE
   - Processus et threads
   - Système de Fichiers
   - Entrée/Sortie
   - Plages d'adressage
   - Protection
   - L'interprète des commandes
5. Appels Système
   - Interruptions
6. Structure des SE
7. Programmation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

Structure des SE

Catégories de SE

- Systèmes monolithiques
- Micro-noyau
- Autres





Systèmes monolithiques

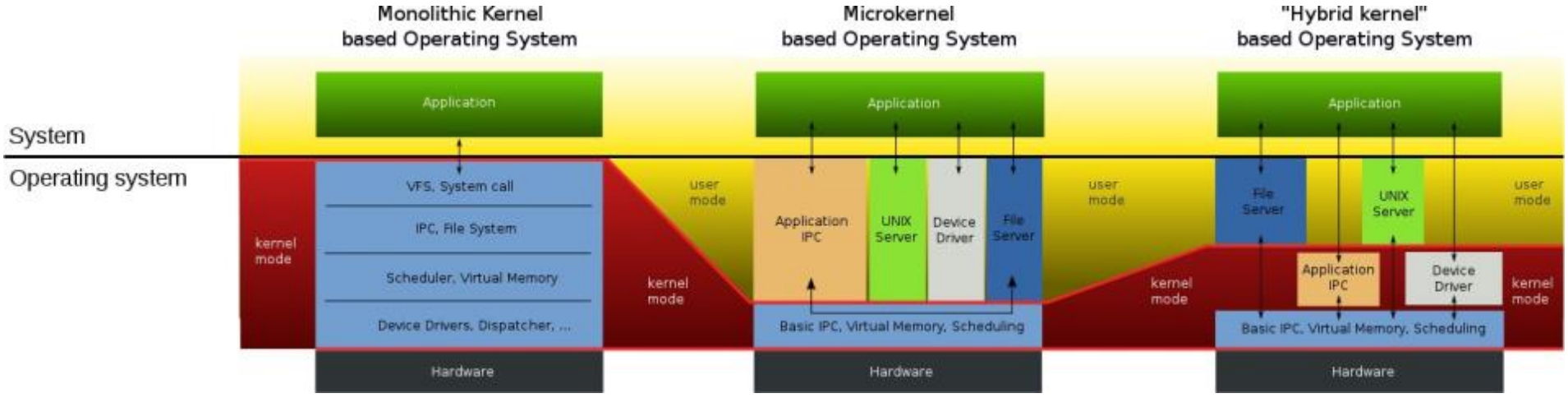
- Tout le SE s'exécute comme **un seul programme** en mode kernel.
- Le SE est un **ensemble de procédures** compilés dans un seule programme exécutable (et assez grand).
- Chaque procédure dans le système peut appeler les autres sans restrictions. Inconvénients :
  - Système peu maniable, difficile 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 son 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

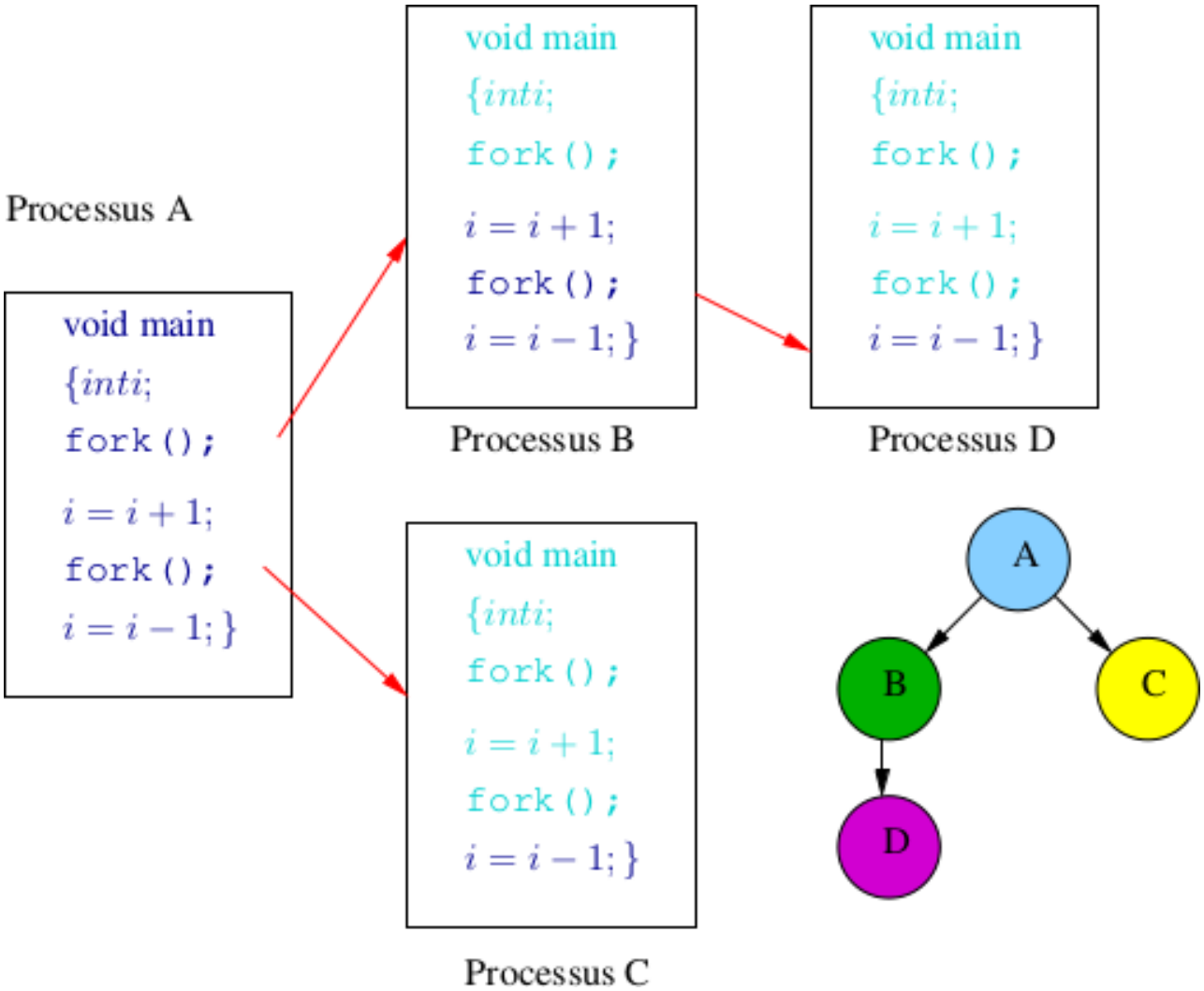
Les processus

- 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

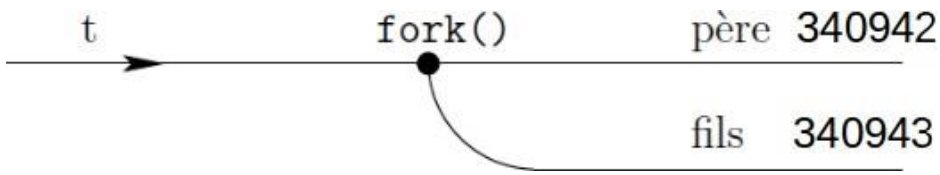
- Appel à l'appel système – **fonction** `fork()` **de C**. Création par clonage (mitose) : un processus (le père) demande, en appelant la primitive `fork()`, la création dynamique d'un nouveau processus (le fils). Le fils s'exécute ensuite de façon concurrente avec le père.
- `pid_t fork (void)`
  - 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

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

int main()
{
   pid_t p;

   p = fork();

   switch(p) {
    case 0 :
      printf("Je suis le fils : mon PID est %d et mon PPID est %d\n",getpid(),getppid());
    break;

    case -1 :
      perror("Erreur de creation de processus avec fork");
    break;

    default :
      printf("Je suis le pere : mon PID est %d et mon PPID est %d\n",getpid(),getppid());
    break;
   }
 return 0;
}
```



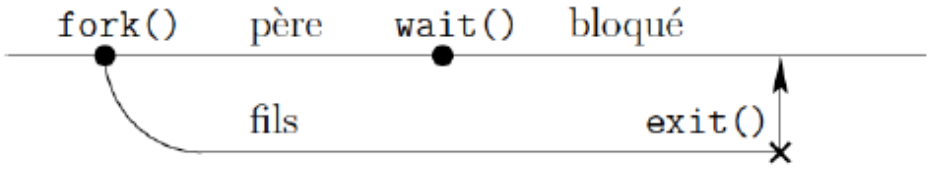
Compilation : `gcc -o useoffork useoffork.c`

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 `init (processus de PID = 0)`
  - `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 avec de la synchronisation

```c
#include <unistd.h>     /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <stdio.h>      /* Input/Output */
#include <sys/wait.h>  /* Wait for Process Termination */
#include <stdlib.h>     /* General Utilities */

int main()
{
    int retval; int status;

    pid_t p = fork();

    if (p == 0) {
      sleep(1); /* sleep for 1 second */
      printf("CHILD: Enter an exit value (0 to 255): ");
      scanf("%d", &retval);
      exit(retval);

    } else if (p > 0) {
        printf("PARENT: I will wait for my child to exit.\n");
        wait(&status);
        printf("PARENT: Child exit code is: %d\n", WEXITSTATUS(status));
        exit(0);

    } else { exit(-1); }
}
```



Compilation : `gcc -o useofwait useofwait.c`

Code equipe TEAMS :

0pjezb6