E4FI : Systèmes d'exploitation

Introduction

Damien MASSON
http://esiee.fr/~massond/

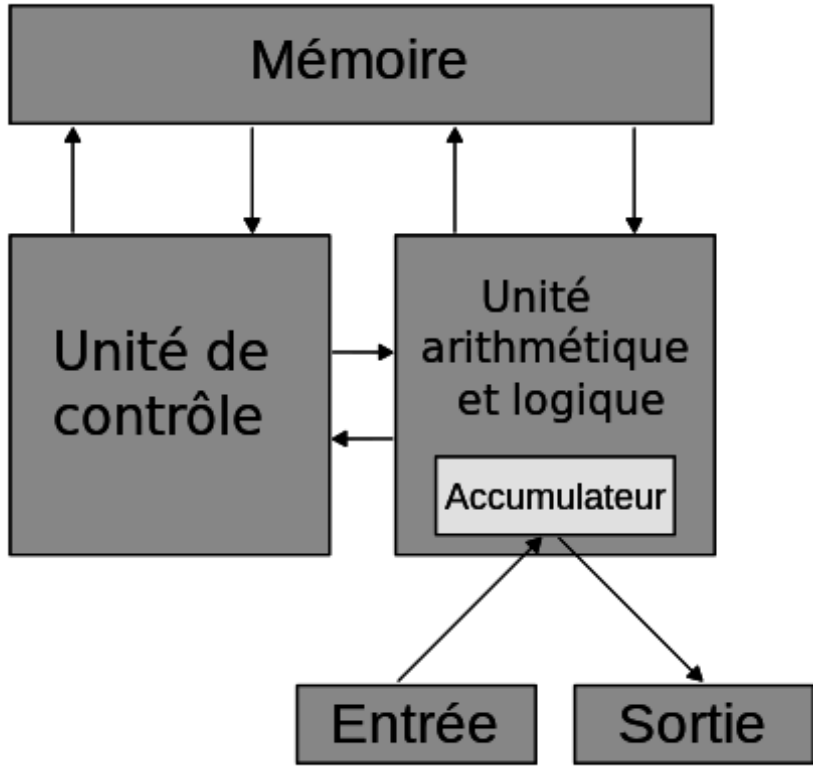
Dernière modification: 13 avril 2026

Informations pratiques

- Damien Masson – Bureau 4206 – `http://www.esiee.fr/~massond`
- Les transparents ne contiennent pas tout : **prendre des notes**
- N'hésitez pas à poser des questions !

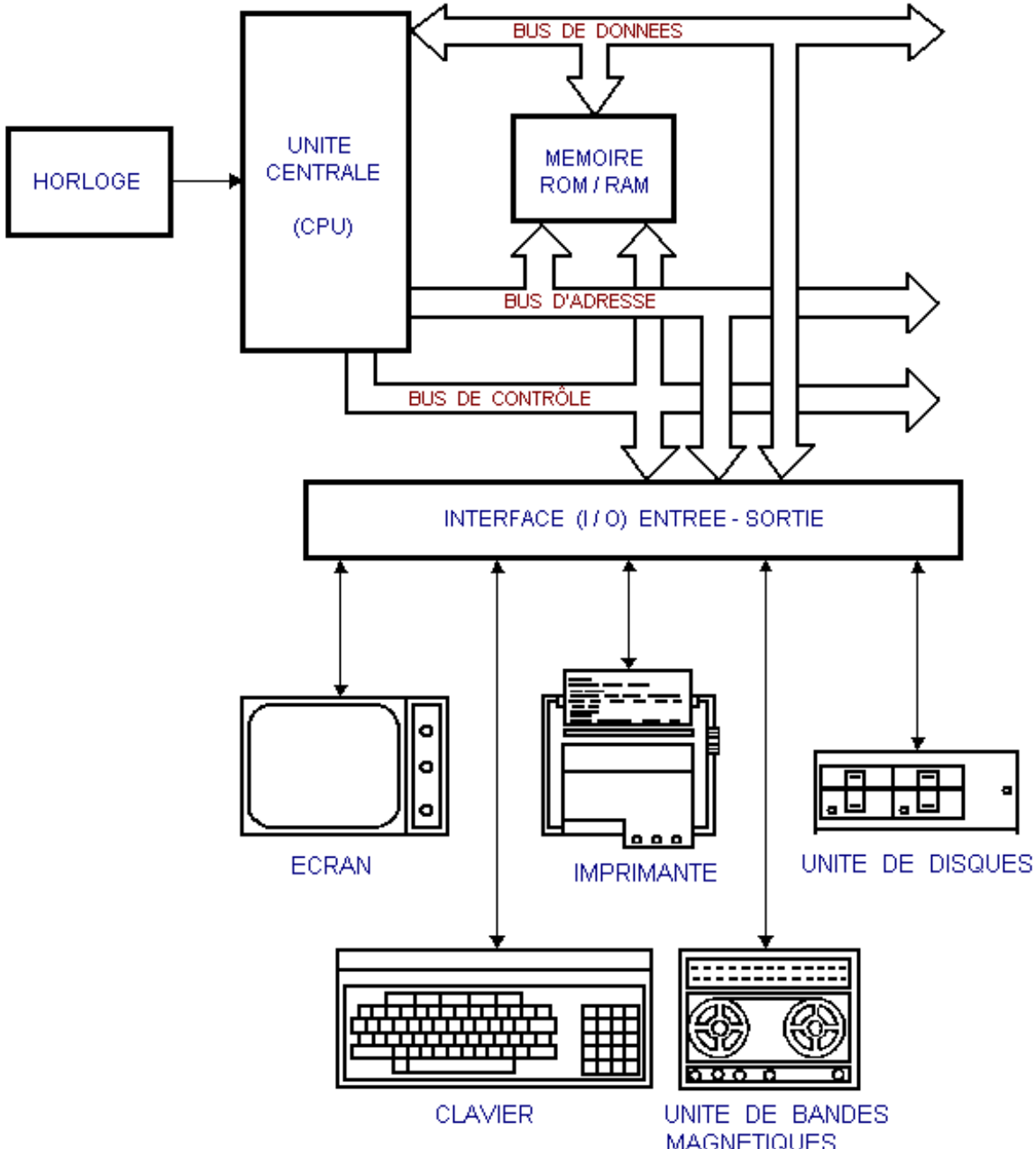
- **Notations :** TP non élalués, mais évaluation finale avec des exo proches de ceux des TP

Modèle de Von Neumann (1903 – 1957)



- L'ordinateur est un automate qui exécute des instructions simples.
- Séparation claire entre l'unité de traitement (CPU) et le stockage (Mémoire).

Schéma synoptique d'un micro-ordinateur



La Mémoire et les Périphériques

La Mémoire centrale

- Contient les données **et** les instructions.
- Grand tableau de "mots binaires" (octets), adressé par des entiers.
- Rapide, mais volatile.

Les périphériques

- Communiquent avec le CPU à travers un bus.
- Très, très lents par rapport au CPU.
- Communication asynchrone, gérée par des **interruptions**.

Exemple théorique : Un programme simple

Imaginons un programme décrit en langage humain :

1. Charge dans le registre A le nombre situé dans la case mémoire 20.
2. Teste le contenu de A : si 0, va à l'instruction 6 (sinon continue).
3. Additionne au registre A le nombre situé dans la case mémoire 21.
4. Copie le contenu du registre A dans la case mémoire 22.
5. Imprime le contenu de la case mémoire 22.
6. Fin.

Instruction

Une instruction machine est la plus petite unité d'action compréhensible par le CPU. Elle contient généralement :

- **Le code opération (Opcode) :** L'action à effectuer (ex : ADD, LOAD, JUMP).
- **Les opérandes :** Les cibles ou sources de l'action. Ce peuvent être :
  - Des registres internes au CPU (ex : `R1`).
  - Des valeurs immédiates (ex : le nombre `42`).
  - Des adresses mémoire (ex : la case `104`).
- **Les drapeaux (Flags) :** Des bits de statut mis à jour dans un registre spécial après l'instruction (ex : "le dernier résultat était zéro", "il y a eu une retenue").

Codage de l'instruction (16 bits)

Format d'une instruction machine :

bits 0 à 4	**bit 5**	**bit 6**	**bits 7 à 15**
Code opération	1er reg. (0 :R, 1 :A)	2e reg. (0 :R, 1 :A)	Adresse mémoire

Exemples de codes opérations (5 bits) :

Instruction	**Code**	**Mnémonique**
Chargement mémoire vers registre	00001	LOAD
Copie registre vers mémoire	00010	STORE
Addition mémoire à registre	00011	ADD
Imprimer mémoire	00100	PRINT
Test registre et branchement si 0	00101	BZ
Fin	00110	END

Traduction en langage machine

Nos 6 instructions se codent donc ainsi :

1. Charge dans le registre A la case 20 $\rightarrow$ `00001 1 0 000010100`
2. Teste A, si 0 va à l'instruction 6 $\rightarrow$ `00101 1 0 000000101`
3. Additionne à A la case 21 $\rightarrow$ `00011 1 0 000010101`
4. Copie A dans la case 22 $\rightarrow$ `00010 1 0 000010110`
5. Imprime la case 22 $\rightarrow$ `00100 1 0 000010110`
6. Fin $\rightarrow$ `00110 0 0 000000000`

Le code final

Une fois chargé en mémoire, débarrassé des espaces et retours à la ligne, le processeur voit ceci :

```
0000110000010100001011000000010100011100000101010001010000010110001001000001011000110000000000000
```

Et si...

- et si on change de type de processeur ?
- et si on déplace les données en mémoire ?
- et si un autre programme tourne en même temps ?

L'empreinte mémoire d'un programme

Quand on lance un programme, il est copié du disque vers la RAM. Son espace est structuré de manière très précise :

Les 4 grandes zones

1. **Le Code (Text) :** Les instructions binaires à exécuter (en lecture seule).
2. **Les Données (Static) :** Les variables globales allouées au démarrage.
3. **Le Tas (Heap) :** La mémoire demandée dynamiquement pendant l'exécution (ex : `malloc` en C). Grandit vers le bas (ou vers le haut selon l'architecture).
4. **La Pile (Stack) :** Stocke les variables locales et l'historique des appels de fonctions. Grandit dans la direction opposée au Tas.

Note : Pour sauvegarder l'état complet de ce programme, il suffit de sauvegarder le contenu des registres du CPU !

Qu'est-ce qu'un périphérique ?

Les périphériques (clavier, disque, carte réseau, écran) permettent à l'ordinateur de communiquer avec le monde extérieur.

- Ils sont connectés au CPU et à la RAM via un **Bus**.
- Ils possèdent souvent leur propre contrôleur (un mini-processeur dédié).
- **Le grand défi :** Ils sont infiniment plus lents que le CPU (des millisecondes contre des nanosecondes).

Le problème de la synchronisation

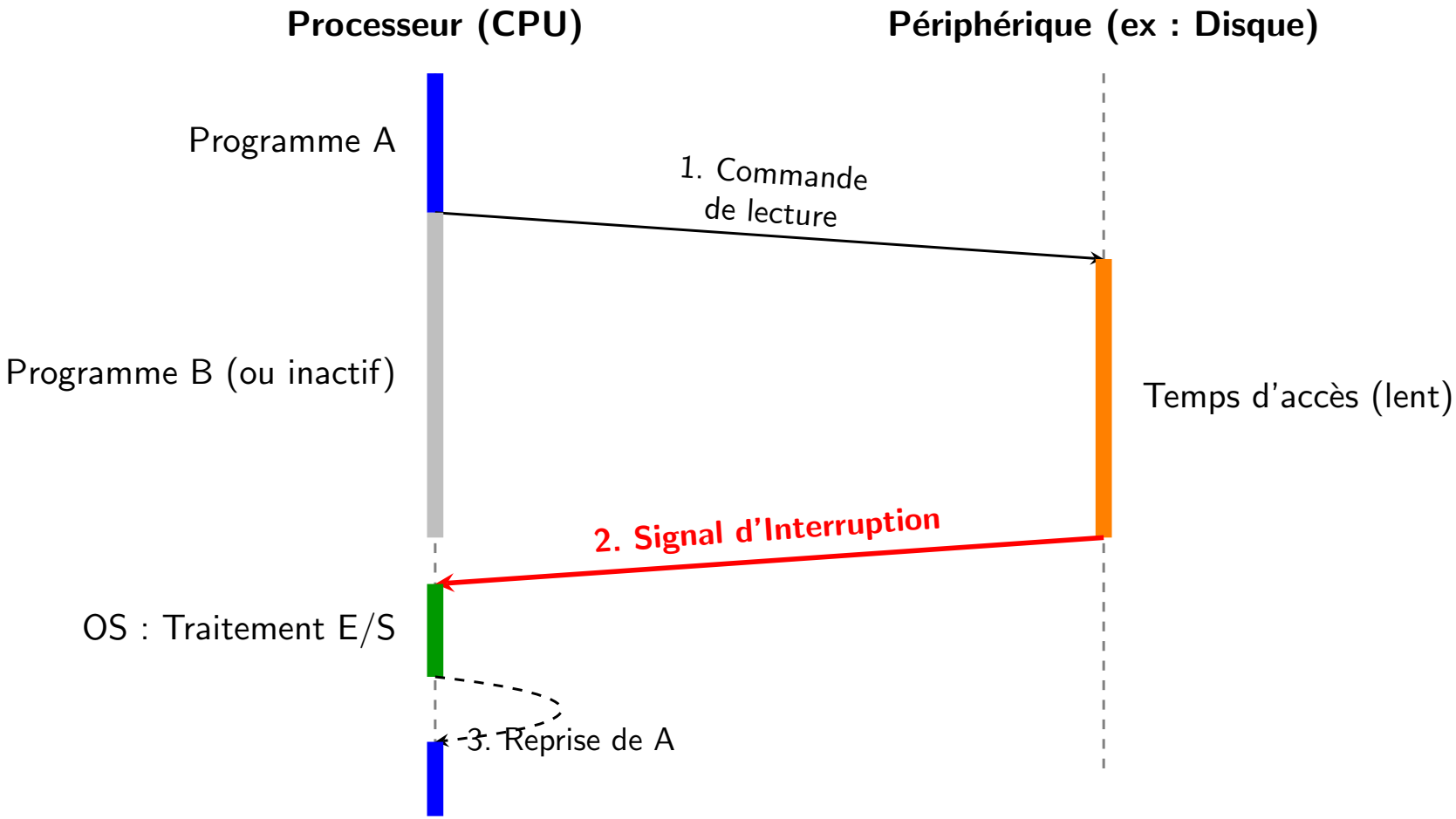
Comment le CPU sait-il qu'un disque dur a fini de lire un fichier ? S'il attend sans rien faire (*attente active* ou *polling*), il gaspille des milliards de cycles de calcul.

Les Interruptions : Le réveil du CPU

Pour éviter le gaspillage, on utilise un mécanisme matériel asynchrone : **l'interruption**.

1. Le CPU envoie une commande au périphérique ("Lis ce secteur du disque").
2. Le CPU **passe à autre chose** (ex : il exécute un autre programme).
3. Quand le périphérique a terminé, il envoie un signal électrique direct au CPU : c'est l'interruption.
4. Le CPU met en pause son travail actuel, traite l'interruption (récupère la donnée), puis reprend là où il s'était arrêté.

Schéma temporel d'une interruption



L'apparition de l'OS : Répondre à un besoin

Sans système d'exploitation, il faut résoudre manuellement :

- Comment démarrer un programme ?
- Comment éviter que le CPU n'ait rien à faire pendant l'attente d'un périphérique ?
- Comment s'assurer qu'un programme bogué n'endommage pas le matériel ?
- Comment partager les ressources (CPU, mémoire, disque) entre plusieurs programmes ?

Solution : Un programme spécial qui tourne en permanence et gère tout cela $\rightarrow$ **Le Système d'Exploitation (OS)**.

Les deux grands rôles de l'OS

1. Une Machine Virtuelle (Rôle d'abstraction)

Il masque la complexité et l'hétérogénéité du matériel. Il offre aux autres programmes des services de haut niveau (ex : "Ouvre ce fichier", "Affiche ce texte") de façon transparente.

2. Un Gestionnaire de Ressources (Rôle d'arbitre)

Il orchestre la concurrence. Il gère l'allocation du CPU (ordonnancement), de la mémoire, et gère les conflits d'accès et la sécurité.

La séparation des privilèges

Pour protéger le système, le CPU fonctionne généralement avec deux niveaux de privilèges :

- **Mode Utilisateur (User mode) :** Non protégé. Les applications classiques s'y exécutent. Accès limité au matériel.
- **Mode Noyau (Kernel mode) :** Protégé. L'OS s'y exécute. Accès total au matériel et aux instructions critiques.

Appels Systèmes (System Calls)

Comment un programme en mode utilisateur fait-il pour lire un fichier sur le disque ?

- Il ne peut pas le faire directement (protection).
- Il utilise un **Appel Système**. C'est une porte sécurisée vers le noyau.

Types d'appels systèmes

- Contrôle des processus (fork, exec, exit).
- Gestion des fichiers (open, read, write).
- Communication (pipes, sockets).
- Gestion de l'information (heure système).

Les sous-systèmes de l'OS

Un OS moderne est découpé en gestionnaires spécialisés :

- Gestion des processus (Ordonnancement).
- Gestion de la mémoire (RAM et virtuelle).
- Gestion des fichiers (Système de fichiers).
- Gestion des Entrées/Sorties (Pilotes/Drivers).
- Réseau (Pile TCP/IP).
- Protection et Sécurité.

Gestion des Processus

- **Définition :** Un processus est un programme *en cours d'exécution*.
- Le CPU passe très rapidement d'un processus à l'autre pour donner l'illusion de la simultanéité (Multitâche).

Responsabilités de l'OS :

- Création et destruction.
- Arrêt (mise en pause) et reprise.
- Fournir les mécanismes de synchronisation et de communication.

Gestion de la mémoire et du stockage

Mémoire Principale (RAM)

L'OS garde la trace des zones utilisées, décide qui charger en mémoire, et isole l'espace de chaque processus pour éviter qu'ils ne se piratent entre eux.

Mémoire Secondaire (Disque)

La RAM est volatile. L'OS gère l'espace sur les disques, planifie les accès (pour optimiser les temps de lecture) et maintient le **Système de Fichiers** (répertoires, droits d'accès).

Classification des systèmes d'exploitation

- **Mono/Multi tâches :** Capacité à exécuter plusieurs processus (Windows, Linux, macOS sont multi tâches).
- **Mono/Multi utilisateurs :** Capacité à isoler les environnements de plusieurs utilisateurs simultanés.
- **Centralisés vs Répartis :** L'OS gère-t-il une seule machine matérielle ou distribue-t-il la charge sur un réseau (grappe/cluster) de manière transparente ?
- **Fermés (Propriétaires) vs Ouverts (Libres) :** Accès au code source (ex : Windows/MacOS vs Linux/GNU).

Architectures du Noyau (Kernel)

Noyau Monolithique (ex : Linux, UNIX traditionnels)

Tout (ordonnanceur, pilotes, système de fichiers) tourne dans le même espace noyau. Rapide, mais un pilote bogué peut faire planter toute la machine.

Micro-noyau (Microkernel)

Le noyau ne gère que le strict minimum (communication inter-processus, CPU de base). Le reste (système de fichiers, pilotes) tourne en espace utilisateur.

- **Avantages :** Plus stable, plus sécurisé, facile à étendre.
- **Inconvénients :** Plus lent à cause des messages incessants entre modules.

Le cas pratique de Linux

Classification stricte

Linux est fondamentalement un **noyau monolithique**. Tous les services s'exécutent dans le même espace mémoire protégé : le **Mode Noyau**.

Un monolithe "modulaire"

Pour réduire la taille du noyau, Linux utilise des **Modules Chargeables Dynamiquement (LKM)**.

- On peut brancher une clé USB, l'OS charge le pilote à la volée.
- **Mais attention :** Une fois chargé, le module tourne en mode noyau. Si le pilote de la clé USB a un bug fatal, il peut provoquer un *Kernel Panic*.

Dans un vrai micro-noyau (comme Minix), le pilote de la clé USB tournerait en mode utilisateur. S'il plante, seul le pilote redémarre, pas l'OS.

Machines Virtuelles

- L'OS hyperviseur traite le matériel et un noyau invité... comme si c'était du matériel nu.
- Permet de faire tourner Windows, Linux et macOS en même temps sur la même machine physique.
- Fournit une isolation parfaite et une réplication facile (très utilisé dans le Cloud computing).

Plan des prochaines séances

- Introduction (cette séance)
- Processus et ordonnancement (Comment l'OS partage le CPU)
- Communication inter-processus et Threads
- Synchronisation (Gérer les accès concurrents)
- Gestion de la mémoire
- Mémoire Virtuelle (L'illusion de la mémoire infinie)
- Systèmes de fichiers (Comment sont stockées les données)