

# Synthèse LINFO1252

Quentin Bodart

Q1 2024-2025

## Contents

<b>1</b>	<b>Le système informatique et le rôle du système d'exploitation</b>	<b>2</b>
1.1	Fondamentaux . . . . .	2
1.2	Architecture de von Neumann . . . . .	2
1.3	Fonctionnement d'un système informatique . . . . .	3
1.4	Traitement d'une interruption . . . . .	3
1.5	Accès direct à la mémoire . . . . .	3
1.6	Système informatique complet . . . . .	4
1.7	Rôle du système d'exploitation . . . . .	4
1.8	Virtualisation . . . . .	4
1.9	Séparation entre mécanisme et politique . . . . .	4
1.10	Modes d'exécution . . . . .	5
1.11	Appel système . . . . .	5
<b>2</b>	<b>Utilisation de la ligne de commande</b>	<b>5</b>
2.1	Utilitaires UNIX . . . . .	6
2.2	Shell / Interpréteur de commandes . . . . .	6
2.3	Flux et redirections . . . . .	6
2.4	Scripts . . . . .	7

## Objectifs du cours

- utiliser et comprendre les systèmes informatiques (i.p. GNU/Linux)
- utiliser les services fournis par les SE (systèmes d'exploitation)
- design et mise en oeuvre d'un SE

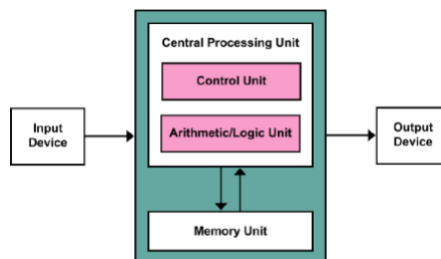
## 1 Le système informatique et le rôle du système d'exploitation

**Syllabus :** <https://sites.uclouvain.be/SystInfo/notes/Theorie/intro.html>

### 1.1 Fondamentaux

- Composants :
  - CPU / Processeur
  - Mémoire Principale (RAM)
  - Dispositifs d'entrée/sortie (y.c. de stockage)
- Fonctionnement d'un CPU
  - Lire / écrire en mémoire vers / depuis des registres
  - Opérations (calculs, comparaisons) sur ces registres
- Jeux d'instructions
  - x86\_64 (PC, anciens Mac)
  - ARM A64 (Raspberry PI, iPhone, nouveaux Mac M1-3)

### 1.2 Architecture de von Neumann

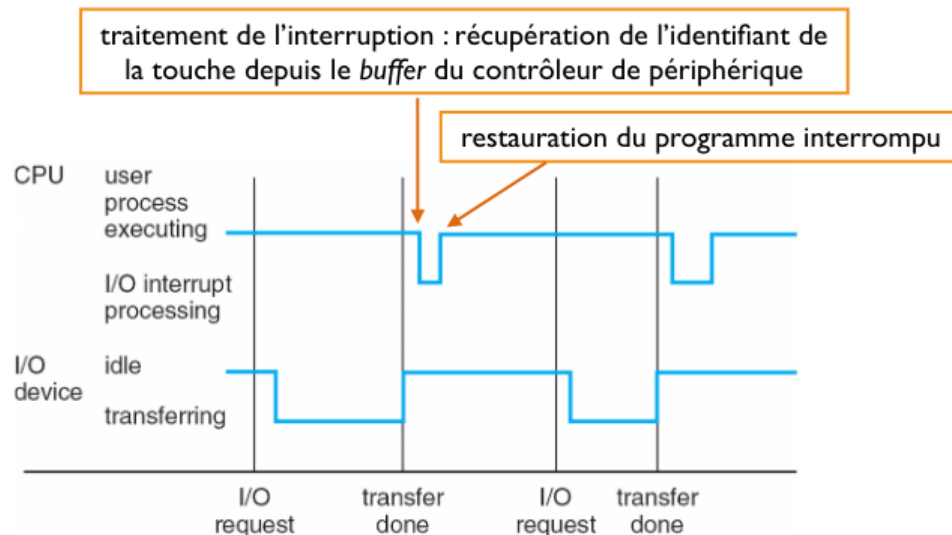


### 1.3 Fonctionnement d'un système informatique

- La représentation des données se fait en **binaire**.
- Les opérations d'entrée-sortie se déroulent de manière concurrente.
- Il y a des contrôleurs distincts contrôlant chacun un type de périphérique.
- Chaque contrôleur possède une mémoire dédiée (buffer)
- Le processeur doit déplacer des données depuis/vers la mémoire principale depuis/vers ces buffers dédiés
- Le processeur suit un "fil" continu d'instructions
- Le contrôleur de périphérique annonce au processeur la fin d'une opération d'entrée/sortie en générant une interruption (signal électrique à destination du processeur)

### 1.4 Traitement d'une interruption

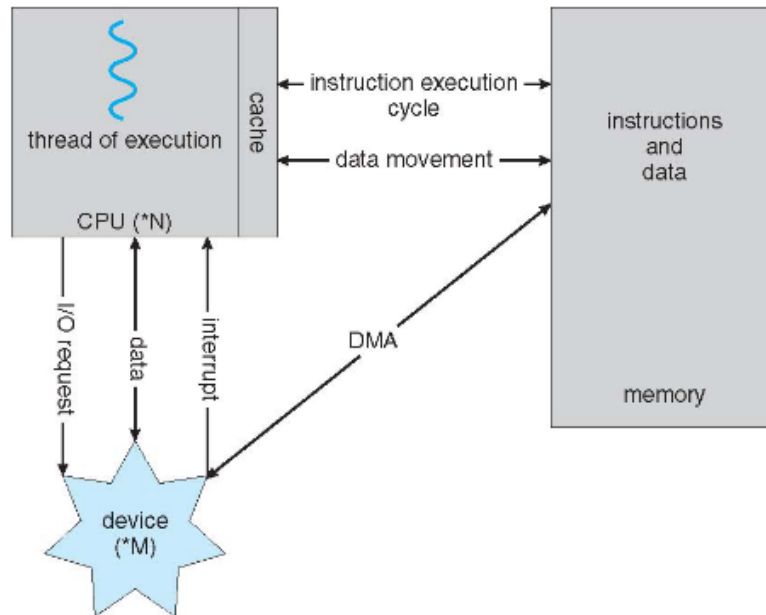
Le processeur interrompt le fil d'exécution d'instructions courant et transfère le contrôle du processeur à une routine de traitement. Cette même routine détermine la source de l'interruption, puis restaure l'état du processeur et reprend le processus :



### 1.5 Accès direct à la mémoire

Direct Memory Access (DMA) désigne le fait de ne pas faire une interruption à chaque octet lu depuis un disque dur. Une interruption est tout de même faite à la fin du transfert d'un **bloc**.

## 1.6 Système informatique complet



## 1.7 Rôle du système d'exploitation

Programmer directement au-dessus du matériel, gérer les interruption, etc... serait une trop grosse tâche pour le programmeur.

3 rôles principaux

- Rendre l'utilisation et le développement d'applications plus simple et plus universel (portable d'une machine à une autre)
- Permettre une utilisation plus efficace des ressources
- Assurer l'intégrité des données et des programmes entre eux (e.g., un programme crash mais pas le système)

## 1.8 Virtualisation

Le système d'exploitation **virtualise** les ressources matérielles afin de fonctionner de la même manière sur des systèmes avec des ressources et composants fort différents. Chaque SE doit trouver un compromis entre abstraction et efficacité !

## 1.9 Séparation entre mécanisme et politique

- Un mécanisme permet le partage de temps

- Une politique arbitre entre les processus pouvant s'exécuter et le(s) processeur(s) disponibles

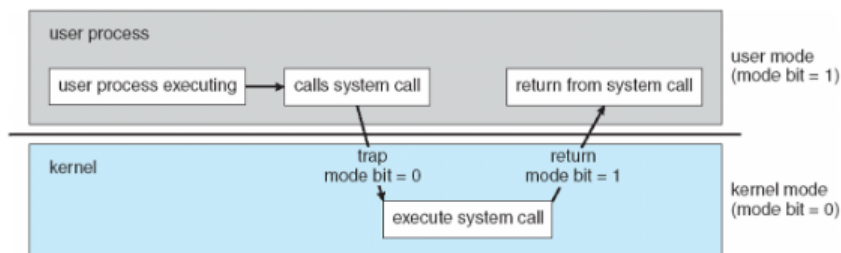
On peut définir des politiques d'ordonnancement différentes selon les contextes, mais sur la base du même mécanisme.

### 1.10 Modes d'exécution

- mode utilisateur : programme utilisant les abstractions fournies par le SE ; certaines instructions sont interdites, comme par exemple:
  - Accès à une zone mémoire non-autorisée (SegFault)
  - De manière générale, toutes les instructions permettant de changer la configuration matérielle du système, comme la configuration ou la désactivation des interruptions
- mode protégé : utilisé par le noyau du SE, toutes les instructions sont autorisées
- L'utilisation de fonctionnalités du SE par un processus utilisateur nécessite de passer d'un mode à l'autre : **appel système**

### 1.11 Appel système

- Un appel système permet à un processus utilisateur d'invoquer une fonctionnalité du SE
- Le processeur interrompt le processus, passe en mode protégé, et branche vers le point d'entrée unique du noyau :



## 2 Utilisation de la ligne de commande

Syllabus : <https://sites.uclouvain.be/SystInfo/notes/Theorie/shell/shell.html>

## 2.1 Utilitaires UNIX

La philosophie lors de la création des utilitaires UNIX était de créer des outils les plus simples possible, donc d'avoir une seule tâche par outil :

### Quelques utilitaires standard

Utilitaire	Fonction
<code>cat</code>	lire/afficher le contenu d'un fichier ex : <code>cat fichier.txt</code>
<code>echo</code>	afficher une chaîne de caractères passée en argument, ex : <code>echo "Bonjour Monde"</code>
<code>head / tail</code>	affiche le début resp. la fin d'un fichier ex : <code>tail errors.log</code>
<code>wc</code>	compte le nombre de caractères / de lignes d'un fichier. ex : <code>wc -l students.dat</code>
<code>sort</code>	trie un fichier. ex : <code>sort -n -r scores.txt</code>
<code>uniq</code>	extraire les lignes uniques ou dupliquées d'un fichier <b>trié</b> fourni en argument. ex : <code>uniq -d students.dat</code>

Afin d'en savoir plus sur

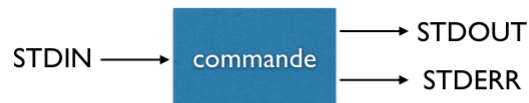
unecommande, il suffit d'utiliser l'utilitaire 'man'.

## 2.2 Shell / Interpréteur de commandes

Rend possible l'interaction avec le SE. Il en existe plusieurs, mais le principal est 'bash'. Il est toujours complémentaire à une **interface graphique**.

## 2.3 Flux et redirections

### Flux standards et redirections



- 3 flux standards (1 entrée, 2 sorties)
- Redirections permettent de combiner des commandes en utilisant des fichiers intermédiaires

<code>&lt; file</code>	redirige le contenu de file vers STDIN
<code>&gt; file</code>	redirige STDOUT vers file (si le fichier n'existe pas, il est créé, si il existe déjà son contenu est écrasé)
<code>&gt;&gt; file</code>	redirige STDOUT vers file (si le fichier n'existe pas, il est créé, si il existe déjà le contenu est ajouté à la suite du fichier)
<code>2&gt;&amp;1</code>	redirige STDERR vers STDOUT

Exemple de redirections :

```
$ echo "Un petit texte" | wc -c
15
$ echo "bbbb ccc" >> file.txt
$ echo "aaaaa bbbbb" >> file.txt
$ echo "bbbb ccc" >> file.txt
$ cat file.txt
bbbb ccc
aaaaa bbbbb
bbbb ccc
$ cat file.txt | sort | uniq
aaaaa bbbbb
bbbb ccc
```

## 2.4 Scripts

Un système UNIX peut exécuter du langage machine ou des **langages interprétés**

Un script commence par convention par les symboles `#!/`, référant à l'interpréteur, ici `bin/bash` :

```
#!/bin/bash
echo "Hello, world"
```

Ils peuvent aussi contenir des **variables**:

```
#!/bin/bash
PROG="LINFO"
COURS=1252
# concaténation
echo $PROG$COURS
```

# : commentaire

accès à la valeur de la variable avec `$`

⚠ une variable non déclarée vaut "" (chaîne vide)

⚠ ambiguïté possible ?

milieu = "mi"; echo do\$milieuno  
utiliser des `{ }` pour délimiter le nom de la variable :  
echo do\${milieu}no

et des **conditionnelles** :

## Conditionnelles (if)

⚠ espace important (source de bug commune)

```
#!/bin/bash
# Vérifie si les deux nombres passés en arguments sont égaux
if [ $# -ne 2 ]; then
    echo "Erreur, deux arguments sont nécessaires" > /dev/stderr
    exit 2
fi
if [ $1 -eq $2 ]; then
    echo "Nombres égaux"
else
    echo "Nombres différents"
fi
exit 0
```

⚠ ; ou passage à la ligne (pas les 2)

`$1 -eq $2` est vraie lorsque les deux variables `$1` et `$2` contiennent le même nombre.  
`$1 -lt $2` est vraie lorsque la valeur de la variable `$1` est numériquement strictement inférieure à celle de la variable `$2`  
`$1 -ge $2` est vraie lorsque la valeur de la variable `$1` est numériquement supérieure ou égale à celle de la variable `$2`  
`$s = $t` est vraie lorsque la chaîne de caractères contenue dans la variable `$s` est égale à celle qui est contenue dans la variable `$t`  
`-z $s` est vraie lorsque la chaîne de caractères contenue dans la variable `$s` est vide

et des **boucles for**:

```
#!/bin/bash
# exemple_for.sh
students="Julie Maxime Hakim"
for s in $students; do
    l="wc -l TP1-$s.txt | cut -d' ' -f1"
    echo "Bonjour $s, ton compte rendu de TP comporte $l lignes."
done
```

- `s` prend successivement les valeurs présentes dans la liste d'entrée `$students`
- ``command`` permet d'assigner la sortie d'une commande à une variable
  - que fait la commande composée ci-dessus ?
- boucles `while` et `until` :  
principe similaire, avec une condition d'arrêt booléenne