SIN4U31L Systèmes d'exploitation

I : Introduction

Léo Henry

22 Janvier 2026

amU Faculté des sciences Aix Marseille Université

Organisation

Séances :

- 6 CM
- 4 TD
- 10 TP

Notation : $\frac{2}{3}$ Examen terminal + $\frac{1}{3}$ TP
Seconde session : Maximum de l' Examen terminal et de $\frac{2}{3}$ Examen terminal + $\frac{1}{3}$ TP.

Références

Références :

- Structured Computer Organization (Andrew S. Tanenbaum, Todd Austin)
- Operating Systems Three Easy Pieces (Remzi H. Arpaci-Dusseau Andrea C Arpaci-Dusseau)
- Operating System Concepts (Abraham Silberschatz, Greg Gagne, Peter B. Galvin)

Avec une versoin française :

- Modern Operating Systems (A. Tanenbaum.) Pearson Education International. 4 ème édition. 2015
- Systèmes d'Exploitation (A. Tanenbaum), 3ème édition. Pearson Education. 2008

Pour aller plus loin :

- Foundations of Computer Science C Edition (Alfred V. Aho, Jeffrey D. Ullman)
- Computer Architecture, Sixth Edition A Quantitative Approach (John L. Hennessy, David A. Patterson)

Les figures de ce cours viennent de ces livres, Wikipedia et le cours d'E. Godard

Plan

Qu'est-ce qu'un ordinateur ?

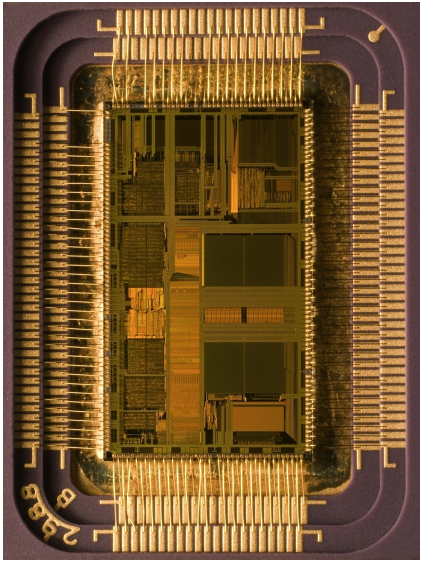
- Machine physique : typiquement basé sur des **micro-processeurs**... mais pas que
- Calculs : opérations logiques / arithmétiques
- Programmes : Un ordinateur est **programmable**, il peut effectuer différents types de calculs.

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

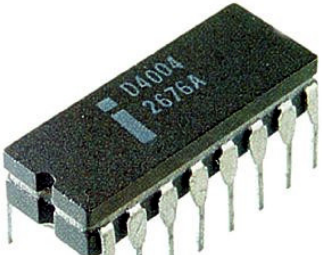
Figure: Computer Architecture Fig. 1.2

Objet physique

Microprocesseur



- Architecture : Nombre et organisation des transistors
- Bits simultannés : de 4 à 32 puis 64
- Jeu d'instructions : Opérations effectuables
- Horloge : Chaque tic est un nouvel instant logiques



Calculs et Programmes

Un ordinateur peut faire des calculs. Au niveau le plus basique ce sont les **instructions machines**, mais pour l'utilisateur ce sont des **applications** et **opérations** de langages de programmation.

Il est possible de **combiner librement** ces opérations dans des **programmes** donnés en entrée à l'ordinateur : le programme peut être défini indépendamment de la machine.

Un ordinateur est une machine physique programmable qui exécute des calculs.

Qu'est-ce qu'un système d'exploitation ?

Dans un ordinateur moderne, on peut avoir **plusieurs programmes** qui doivent être exécutés et qui utilisent des ressources matérielles variées (différents types de mémoire, un clavier, le réseau, un écran...).

Le système d'exploitation est le chef d'orchestre qui organise l'utilisation des ressources pour permettre l'exécution.

Plan

Quel est le rôle du système d'exploitation ?

Rôles clefs :

- **Abstraire** le matériel pour l'utilisateur.
- **Gérer** les ressources matérielles.

Quel est le rôle du système d'exploitation ?

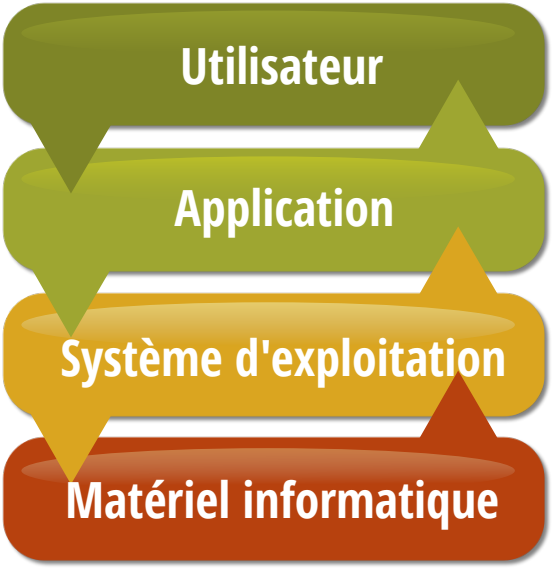
Rôles clefs :

- **Abstraire** le matériel pour l'utilisateur.
- **Gérer** les ressources matérielles.

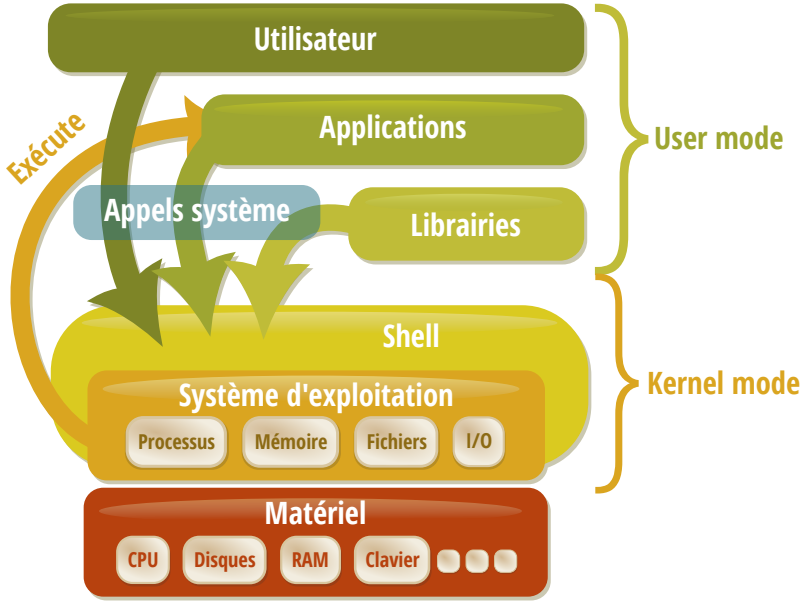
Propriétés désirables :

- **Performance** : charge du système, efficacité de l'utilisation des ressources, dépense énergétique...
- **Isolation** des programmes
- **Protection** des données et du matériel et plus généralement **Sécurité**
- **Fiabilité** : l'OS tourne tout le temps, et des erreurs peuvent être catastrophiques

Abstraction



Abstraction



Gestion du matériel

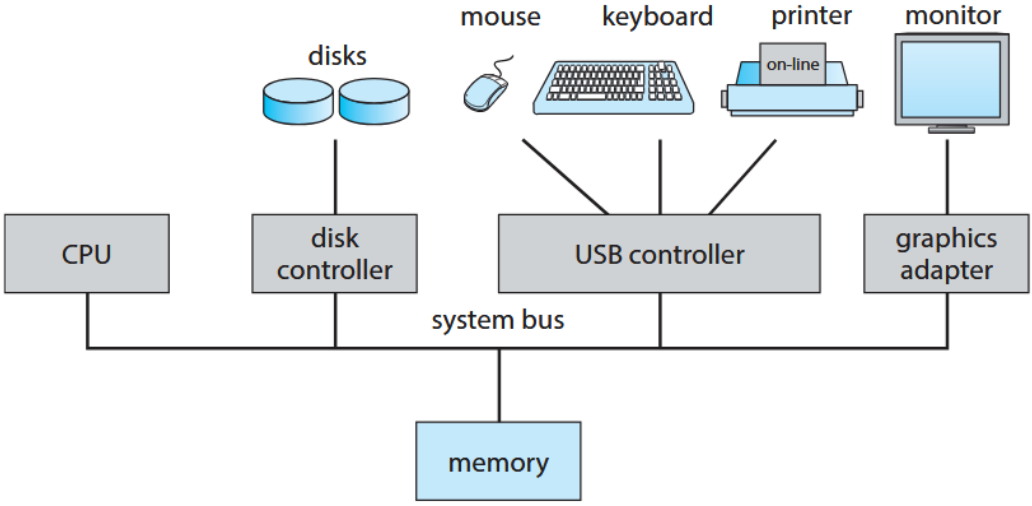


Figure: Operating System Concepts Fig. 1.2

Vue de l'OS

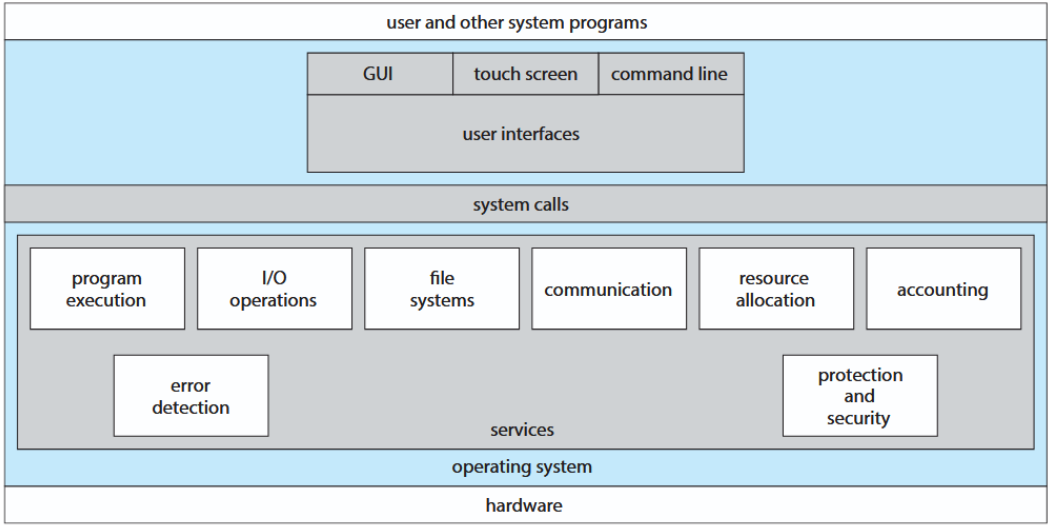


Figure: Operating Systems Concepts Fig. 2.1

Plan

3 clefs du système d'exploitation

- **Virtualisation** Remplacer la ressource physique par une **ressource virtuelle**, plus générique et aisée à manipuler. On construit une **machine virtuelle** accessible par les **appels systèmes**.

3 clefs du système d'exploitation

- **Virtualisation** Remplacer la ressource physique par une **ressource virtuelle**, plus générique et aisée à manipuler. On construit une **machine virtuelle** accessible par les **appels systèmes**.
- **Concurrence** Gérer les problèmes dus à l'**exécution de plusieurs tâches** dans un unique programme (dans notre cas l'OS).

3 clefs du système d'exploitation

- **Virtualisation** Remplacer la ressource physique par une **ressource virtuelle**, plus générique et aisée à manipuler. On construit une **machine virtuelle** accessible par les **appels systèmes**.
- **Concurrence** Gérer les problèmes dus à l'**exécution de plusieurs tâches** dans un unique programme (dans notre cas l'OS).
- **Persistence** Assurer que lors de l'arrêt de l'ordinateur (ou lors d'un bogue), les données persistent.

Virtualisation du CPU

```
1 import sys
2 a = sys.argv[1]
3
4 while True:
5   print(a)
```

```python
import sys
a = sys.argv[1]

while True:
  print(a)
```

Exécution sur 1 cœur

```
> python ./test.py A
A
A
A
^C
```

Rien de très surprenant...

```python
import sys
a = sys.argv[1]

while True:
  print(a)
```

Exécution sur 1 cœur

```
> python ./test.py A
A
A
A
^C
```

Rien de très surprenant...

Exécutions multiples sur 1 cœur

```
> python ./test.py A &
  python ./test.py B &
  python ./test.py C
A
B
C
A
B
C
^C
```

Illusion d'un **processeur virtuel** par processus pour un seul processus réel.

Virtualisation de la mémoire

```
1 p = 0.0
2 print(os.getpid())
3 print("address␣"+str(id(p)
    ))
4 for i in range(3):
5   p+=1
6   time.sleep(0.5)
7   print(p)
```

Virtualisation de la mémoire

```
1 p = 0.0
2 print(os.getpid())
3 print("address␣"+str(id(p)
    ))
4 for i in range(3):
5   p+=1
6   time.sleep(0.5)
7   print(p)
```

Exécution sur 1 cœur

```
> python ./test.py
22364
address 140735142757256
1.0
2.0
3.0
```

Virtualisation de la mémoire

```
1 p = 0.0
2 print(os.getpid())
3 print("address␣"+str(id(p)
    ))
4 for i in range(3):
5   p+=1
6   time.sleep(0.5)
7   print(p)
```

Exécution sur 1 cœur

```
> python ./test.py
22364
address 140735142757256
1.0
2.0
3.0
```

Exécutions multiples sur 1 cœur

```
> python ./test.py &
  python ./test.py
22364
22365
address 140735142757256
address 140735142757256
1.0
1.0
2.0
2.0
3.0
3.0
```

Concurrence

```
def run(letter):
    i = 0
    while i < 20:
        sys.stdout.write(
            letter)
        sys.stdout.flush()
        wait = 0.2
        wait += random.
            randint(1, 60) /
            100
        time.sleep(wait)
        i += 1
```

```
if __name__ == "__main__":
    t1 = Thread(target = run
        ,args=("1",))
    t2 = Thread(target = run
        ,args=("2",))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

Concurrence

```
1 def run(letter):
2   i = 0
3   while i < 20:
4       sys.stdout.write(
            letter)
5       sys.stdout.flush()
6       wait = 0.2
7       wait += random.
            randint(1, 60) /
            100
8       time.sleep(wait)
9       i += 1
```

```
1 if __name__ == "__main__":
2   t1 = Thread(target = run
        ,args=("1",))
3   t2 = Thread(target = run
        ,args=("2",))
4   t1.start()
5   t2.start()
6   t1.join()
7   t2.join()
```

Exécution : 12121221212121112121212121211121221222
mais aussi 12121122221211221121212112221212121121122

Plan

Historique

Une accumulation progressive de bonnes idées.

- Ere de la **programmation par lot** (gestion par un humain, pas d'interaction) : une simple librairie abstrayant les I/O.
- Début de la protection : le **shell**. Introduction des **appels systèmes** et séparation entre user et kernel mode.
- Le mini-ordinateur : gestion de **programmes multiples**, protection de la mémoire, concurrence. Et **UNIX** !
- L'**ordinateur personnel** : un retour de l'âge sombre suivi de nombreuses améliorations.

à Bell Labs, UNIX rassemble les bonnes idées des premiers jours.

- Propose l'interface commande du shell : primitives de **meta-programmation** (ex. tubes / pipes). Permet de **combiner des programmes** !

```
> grep foo file.txt|wc -l
```

- Intègre un compilateur du jeune langage C
- Code libre et largement distribué... menant à une bataille légale.
- UNIX est sauvé par **Linux**, qui reprend les techniques mais recode entièrement l'OS.

Plan

Machine virtuelle et hyperviseur