

Processing en mode actif ⊕ systèmes d'exploitation

Valeur d'accueil et de reconversion en informatique (VARI1)

Daniel Porumbel (dp.cnam@gmail.com)

<http://cedric.cnam.fr/~porumbed/var1/>

1 Méthodes `setup` et `draw` \implies animations

2 Systèmes d'exploitation

Méthode `setup()`

Jusqu'à présent : nous avons directement écrit le code actif (en vrac)

```
println ("Toto");  
ellipse (....)  
faireQuelqueChose ();  
faireAutreChose ();  
...
```

A partir de maintenant : Le code actif fait partie d'une **méthode**,
comme `setup()`. Les instructions sont entourées
de `void nomMéthode() {` et de `}`

Le code est une liste de méthodes ; la première exécutée est :

```
setup() SOUS Processing  
main() SOUS C/C++/Java
```

Méthode `setup()`

Jusqu'à présent : nous avons directement écrit le code actif (en vrac)

```
println ("Toto");  
ellipse (....)  
faireQuelqueChose ();  
faireAutreChose ();  
...
```

A partir de maintenant : Le code actif fait partie d'une **méthode**,
comme `setup` Les instructions sont entourées
de `void nomMéthode() {` et de `}`

Le code est une liste de méthodes ; la première exécutée est :

```
setup() SOUS Processing  
main() SOUS C/C++/Java
```

Méthode `setup()`

Jusqu'à présent : nous avons directement écrit le code actif (en vrac)

```
println ("Toto");  
ellipse (....)  
faireQuelqueChose ();  
faireAutreChose ();  
...
```

A partir de maintenant : Le code actif fait partie d'une **méthode**,
comme `setup` Les instructions sont entourées
de `void nomMéthode() {` et de `}`

Le code est une liste de méthodes ; la première exécutée est :

```
setup() SOUS Processing  
main() SOUS C/C++/Java
```

Mode statique et mode actif

Processing connaît deux modes de travail :

mode actif on déclare plusieurs méthodes

mode statique pas de méthodes, juste des instructions vrac

Impossible de mélanger les deux modes

```
...  
println ( "Toto" );  
void setup () {  
    println ( "Nous_sommes_dans_une_methode" );  
}  
...
```

Mode statique et mode actif

Processing connaît deux modes de travail :

mode actif on déclare plusieurs méthodes

mode statique pas de méthodes, juste des instructions vrac

Impossible de mélanger les deux modes

```
..  
println("Toto");  
void setup(){  
    println("Nous sommes dans une methode");  
}  
...
```

Impossible de déterminer s'il faut afficher
"toto" avant ou après `setup()` ?

La méthode `draw()`

- est appelée de manière **répétitive** par Processing
 - Rappel : `setup()` c'est que pour l'initialisation
- Le nombre d'appels par seconde est géré par `frameRate(..)`
- Une première animation ; des lignes aléatoire répétitives en utilisant `draw()`

```
void setup() {  
    size(600,600);  
    frameRate(1000);  
}  
void draw() {  
    line(random(600),random(600), // pixel x  
         random(600),random(600)); // pixel y  
}
```


La méthode `draw()`

- est appelée de manière **répétitive** par Processing
 - Rappel : `setup()` c'est que pour l'initialisation
- Le nombre d'appels par seconde est géré par `frameRate(..)`
- Une première animation ; des lignes aléatoire répétitives en utilisant `draw()`

```
void setup() {  
    size(600,600);  
    frameRate(1000);  
}  
void draw() {  
    line(random(600),random(600), // pixel x  
         random(600),random(600)); // pixel y  
}
```

Vers une forme d'art abstrait :)

Tester ce code

- Le dernier argument de `fill(...)` indique un niveau de transparence

```
void setup() {  
  size(600,600);  
  frameRate(10);  
}  
void draw() {  
  noStroke();  
  fill(random(255),random(255),  
        random(255),50);  
  rect(random(600),random(600),50,50);  
}
```

Il est possible de déclarer des variables à l'extérieur de toute méthode

Ce sont des variables **globales** que toute méthode peut utiliser

```
int nbAppels ;
void setup() {
    size(600,600);
    frameRate(10);
    nbAppels = 0;
}
void draw() {
    nbAppels = nbAppels + 1;
    if (nbAppels < 50)
        line(random(600), random(600),
            random(600), random(600));
}
```

Utiliser `draw()` comme une boucle implicite

```
int x;  
void setup() {  
    size(500,500);  
    x= 0;  
}  
void draw() {  
    background(200);  
    ellipse(x,x,10,10);  
    x++;           // ou x=x+1  
}
```

Est-il facile de faire rebondir la balle ?

- 1 Méthodes `setup` et `draw` \implies animations
- 2 Systèmes d'exploitation

La vie sans système d'exploitation

- Au tout début, les machines **ne possédaient pas** de système d'exploitation. Il fallait gérer directement le CPU, la mémoire, les périphériques, etc...
 - C'est comme si on devait prendre une décision consciente pour activer chaque muscle dont a besoin pour se déplacer
 - Un programme écrit sur une machine ne pouvait pas tourner sur une autre
- La couche **Système d'Exploitation** permet de libérer les programmeurs de la gestion directe du matériel
 - c'est un peu comme les muscles qui s'activent *machinalement* si on décide de donner la commande de marcher

La vie sans système d'exploitation

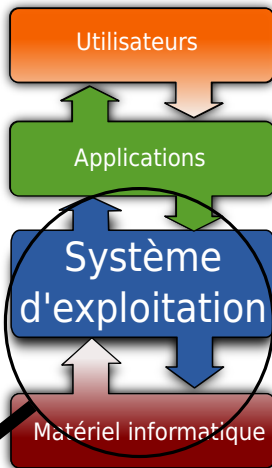
- Au tout début, les machines **ne possédaient pas** de système d'exploitation. Il fallait gérer directement le CPU, la mémoire, les périphériques, etc...
 - C'est comme si on devait prendre une décision consciente pour activer chaque muscle dont a besoin pour se déplacer
 - Un programme écrit sur une machine ne pouvait pas tourner sur une autre
- La couche **Systeme d'Exploitation** permet de libérer les programmeurs de la gestion directe du matériel
 - c'est un peu comme les muscles qui s'activent *machinalement* si on décide de donner la commande de marcher

Rappels couches génériques d'ordinateurs

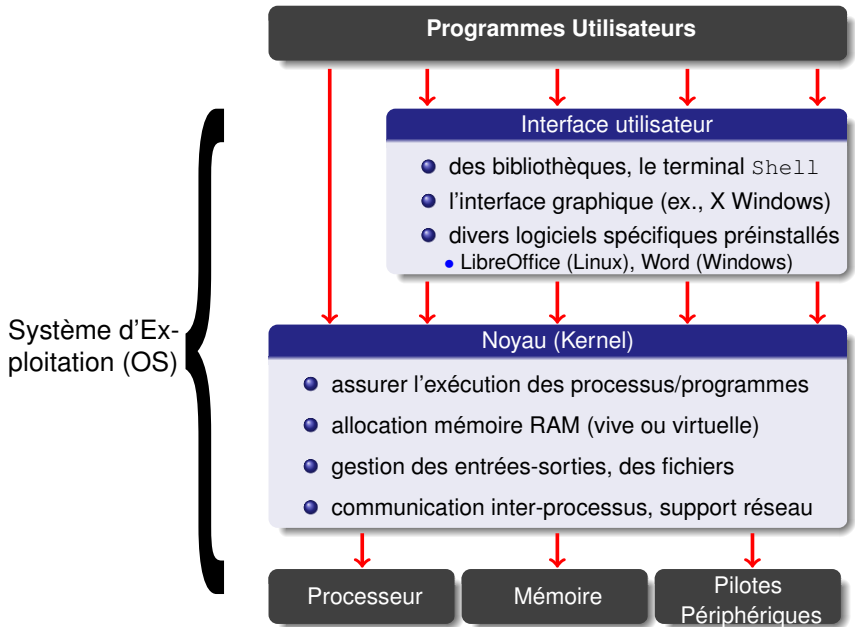
Couche Système d'Exploitation

- gère l'utilisation du matériel par les applications ;
- gestion programmes (multi-tache), communication inter-processus ;
- mémoire, le système de fichiers ;
- interface utilisateur, terminal shell et programmes utilitaires ;
- pilotes (en. : drivers) périphériques

Nous allons zoomer sur cette couche



Composition Système d'Exploitation (OS)



Windows, macOS, iOS et (partiellement) Android

- beaucoup d'aspects secrets, fonctionnalités cachés(?)
- ils peuvent **imposer leurs applications**, car il n'est pas toujours comode de les remplacer (logique de fermeture ?)
 - Difficile d'installer l'application RATP sur Android sans se connecter à un compte Google ? Pour quoi ?
- Android est open source mais Google Play (le gestionnaire d'applis standard) ne l'est pas \implies Google reste encore et toujours maître des lieux sur les installations standard

Windows, macOS, iOS et (partiellement) Android

- beaucoup d'aspects secrets, fonctionnalités cachés(?)
- ils peuvent **imposer leurs applications**, car il n'est pas toujours comode de les remplacer (logique de fermeture?)
 - Difficile d'installer l'application RATP sur Android sans se connecter à un compte Google ? Pour quoi ?
- Android est open source mais Google Play (le gestionnaire d'applis standard) ne l'est pas \implies Google reste encore et toujours maître des lieux sur les installations standard

Windows, macOS, iOS et (partiellement) Android

- beaucoup d'aspects secrets, fonctionnalités cachés(?)
- ils peuvent **imposer leurs applications**, car il n'est pas toujours comode de les remplacer (logique de fermeture ?)
 - Difficile d'installer l'application RATP sur Android sans se connecter à un compte Google ? Pour quoi ?
- parfois compatibles **uniquement** avec des périphériques vendus par **la même entreprise** (Apple ?),
- obsolescence programmée : pas de support/applis pour les vieilles versions
 - tout est très difficiles à réparer (pour favoriser la consommation)
- Android est open source mais Google Play (le gestionnaire d'applis standard) ne l'est pas \implies Google reste encore et toujours maître des lieux sur les installations standard
- Le macOS Darwin (open source) a été abandonné

Systèmes ouverts de type Unix/Linux

- logiciels libres et gratuits, **tout** le code est public (open source), c.à.d, on peut voir ce qui se passe dans la cuisine !
- Philosophie générale Linux : l'OS n'est pas un « plat cuisiné » à ne pas modifier/toucher
 - forte **modularité**, ex., on peut garder un même OS mais changer l'interface graphique (passer de Gnome à KDE)
 - Un OS Linux est souvent très **configurable**, avec la liberté de tout modifier/échanger/personnaliser
- **macOs/iOs**, **Android** et **ChromeOS** utilisent du code d'un **noyau de type Unix** (BSD resp. Linux)

Il est possible de tourner un OS dans un autre à l'aide de logiciels de virtualisation, ex., `virtualbox`

Systèmes ouverts de type Unix/Linux

- logiciels libres et gratuits, **tout** le code est public (open source), c.à.d, on peut voir ce qui se passe dans la cuisine !
- Philosophie générale Linux : l'OS n'est pas un « plat cuisiné » à ne pas modifier/toucher
 - forte **modularité**, ex., on peut garder un même OS mais changer l'interface graphique (passer de Gnome à KDE)
 - Un OS Linux est souvent très **configurable**, avec la liberté de tout modifier/échanger/personnaliser
- **macOs/iOs**, **Android** et **ChromeOS** utilisent du code d'un **noyau de type Unix** (BSD resp. Linux)



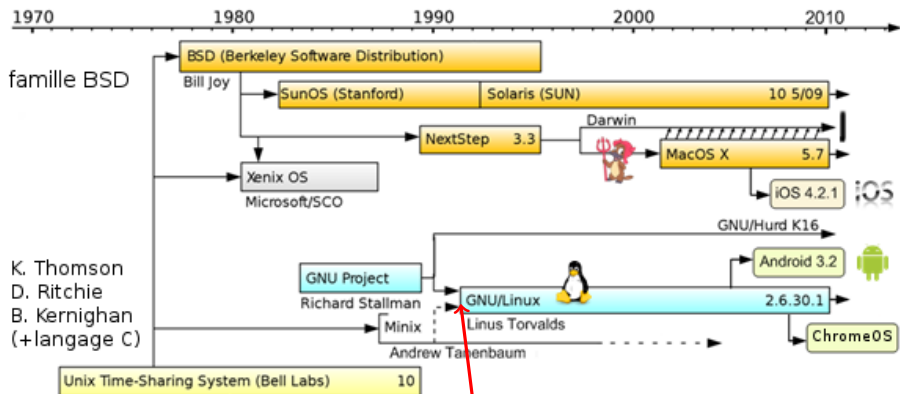
Il est possible de tourner un OS dans un autre à l'aide de logiciels de virtualisation, ex., `virtualbox`

- logiciels libres et gratuits, **tout** le code est public (open source), c.à.d, on peut voir ce qui se passe dans la cuisine !
- Philosophie générale Linux : l'OS n'est pas un « plat cuisiné » à ne pas modifier/toucher
 - forte modularité, ex., on peut garder un même OS mais changer l'interface graphique (passer de Gnome à KDE)
 - Un OS Linux est souvent très **configurable**, avec la liberté de tout modifier/échanger/personnaliser
- **macOs/iOs**, **Android** et **ChromeOS** utilisent du code d'un **noyau de type Unix** (BSD resp. Linux)



Il est possible de tourner un OS dans un autre à l'aide de logiciels de virtualisation, ex., `virtualbox`

Histoire simplifié des OS/noyaux de type Unix/Linux



Le noyau Linux

- créé par Linus Torvalds en 1991 et initialement développé que par ce dernier, et ensuite par son équipe et des contributeurs partout autour du monde (c'est open-source !)
- Il est très petit (quelques MB) et peut tourner sur des machines plus anciennes

Interactions avec l'utilisateur

Tout système (OS) propose des fonctionnalités pour manipuler les fichiers/dossiers et pour lancer des programmes

- MacOS, Windows : logique de base “tout à la souris”
 - double clic sur des icônes, drag-and-drop, etc.
 - clic sur des menus déroulants
- Android (Google), iOS (Apple) : logique “écran tactile”

L'interface graphique masque le fonctionnement de l'OS, et peut rendre l'utilisateur incapable de résoudre des problèmes qui y sont liés

-
- Linux/Unix : à l'origine tout au clavier
 - Un Shell se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal.
 - Plus tard : de nombreuses interfaces graphiques développées par les communautés “open source” (libres)
 - Toutes les couches et composants Linux/Unix peuvent être étudiés séparément : c'est le plus modulaire et transparent OS

⇒ Nous allons continuer notre étude avec Linux

Interactions avec l'utilisateur

Tout système (OS) propose des fonctionnalités pour manipuler les fichiers/dossiers et pour lancer des programmes

- MacOS, Windows : logique de base “tout à la souris”
 - double clic sur des icônes, drag-and-drop, etc.
 - clic sur des menus déroulants
- Android (Google), iOS (Apple) : logique “écran tactile”

- Linux/Unix : à l'origine tout au clavier

- Un `Shell` se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal.
- Plus tard : de nombreuses interfaces graphiques développées par les communautés “open source” (libres)
- Toutes les couches et composants Linux/Unix peuvent être étudiées séparément : c'est le plus modulaire et transparent OS

⇒ Nous allons continuer notre étude avec Linux

Tout système (OS) propose des fonctionnalités pour manipuler les fichiers/dossiers et pour lancer des programmes

- MacOS, Windows : logique de base “tout à la souris”
 - double clic sur des icônes, drag-and-drop, etc.
 - clic sur des menus déroulants
- Android (Google), iOS (Apple) : logique “écran tactile”

- Linux/Unix : à l'origine tout au clavier

- Un `Shell` se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal.
- Plus tard : de nombreuses interfaces graphiques développées par les communautés “open source” (libres)
- Toutes les couches et composants Linux/Unix peuvent être **étudiées séparément** : c'est **le plus modulaire et transparent** OS

⇒ Nous allons continuer notre étude avec Linux

Linux : terminal et interface graphique

Le Shell Unix : un terminal en ligne de commande

- Shell = enveloppe extérieure en anglais. On l'appelle Shell parce que c'est une couche autour du noyau.
- Il cache les détails de l'OS et gère les détails techniques des interactions avec le noyau
- Le premier shell est le *Thompson shell* apparu en 1971

Interfaces graphiques

- De nombreuses versions Open Source (ex., Gnome, KDE)
- La commande `ssh -X AUTREORDI` permet de lancer sur l'écran local des programmes qui tournent sur AUTREORDI
- Un *gestionnaire de fenêtres* se charge de l'affichage/placement des fenêtres (ex., compiz, marco, kwin, IceWm)
- Un serveur X reçoit les commandes de l'interface graphique et du gestionnaire de fenêtres

Linux : terminal et interface graphique

Le Shell Unix : un terminal en ligne de commande

- Shell = enveloppe extérieure en anglais. On l'appelle Shell parce que c'est une couche autour du noyau.
- Il cache les détails de l'OS et gère les détails techniques des interactions avec le noyau
- Le premier shell est le *Thompson shell* apparu en 1971

Interfaces graphiques

- De nombreuses versions Open Source (ex., Gnome, KDE)
- La commande `ssh -X AUTREORDI` permet de lancer sur l'écran local des programmes qui tournent sur AUTREORDI
- Un *gestionnaire de fenêtres* se charge de l'affichage/placement des fenêtres (ex., compiz, marco, kWin, IceWm)
- Un serveur X reçoit les commandes de l'interface graphique et du gestionnaire de fenêtres

Le Shell Unix : un terminal en ligne de commande

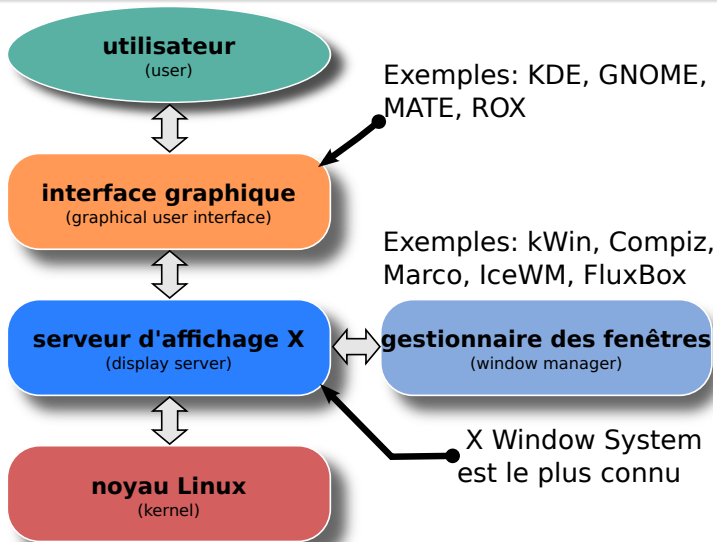
- Shell = enveloppe extérieure en anglais. On l'appelle Shell parce que c'est une couche autour du noyau.
- Il cache les détails de l'OS et gère les détails techniques des interactions avec le noyau
- Le premier shell est le *Thompson shell* apparu en 1971

Interfaces graphiques

- De nombreuses versions Open Source (ex., Gnome, KDE)
- La commande `ssh -X AUTREORDI` permet de lancer sur l'écran local des programmes qui tournent sur AUTREORDI
- Un *gestionnaire de fenêtres* se charge de l'affichage/placement des fenêtres (ex., compiz, marco, kWin, IceWm)
- Un serveur X reçoit les commandes de l'interface graphique et du gestionnaire de fenêtres

Les couches graphiques de Linux reporté

Image due à fr.wikipedia.org/wiki/IceWM



Tout est configurable ! Toutes les combinaisons sont possibles : KDE-Compiz, Gnome-IceWM, Mate-Marco, Rox-IceWM,...