Ordinateur, système d'exploitation

2014-09-02

1 Système d'exploitation

Machine de von Neumann : seul **un** programme s'exécute.
Besoins utilisateurs : plein de programmes.
Système d'exploitation (*Operating System*) : **le** programme chargé au démarrage et orchestrant les autres programmes.

1.1 Responsabilités du SE

Servir :

- Multitâche (illusion)
- Organisation DD (répertoires, fichiers)
- Lancement des applications

Protéger :

- Gestion des utilisateurs
- Contrôle d'accès (DD, réseau, RAM, CPU)

1.2 OS existants

1.2.1 Le plus connu

Famille des systèmes Microsoft Windows. Selon Wikipedia (article *Microsoft*, le 31 août 2013) :

> *La stratégie commerciale de Microsoft, reposant sur la vente liée, lui a permis de diffuser son système d'exploitation Windows sur la grande majorité des ordinateurs de bureau.*

Condamnations par la commission européenne (497×10^6€ en 2004, 280×10^6€, 899×10^6 € en 2008)

1.2.2 La famille régnante : les dérivés d'Unix

Fiables, robustes, assez sécurisés, adaptables aux besoins.
Linux hégémonique :

— 90% des calculateurs du TOP 500

— majoritaire sur les serveurs web

— (probablement) majoritaire sur les box ADSL (Freebox, Livebox, Neufbox et Bewan iBox)

1.3 Le Multitâche, une illusion

Applications : stockées sur le DD.
Interruption matérielle : Dispositif interrompant le programme en cours et donnant la main au système d'exploitation lorsque survient un événement externe ou à une date donnée.
Le système d'exploitation :

— Charge en mémoire vive les applications à exécuter (on parle de *tâches* ou de `processus`).

— Donne la main à une application.

— Au bout de quelques millisecondes survient une interruption.

— Le système reprend la main, traite l'événement, redonne la main à une autre tâche.

1.4 Système de fichiers

— Données organisées par fichiers :

  — Programmes (et leurs données) : $\approx 10^5$ fichiers sur mon PC

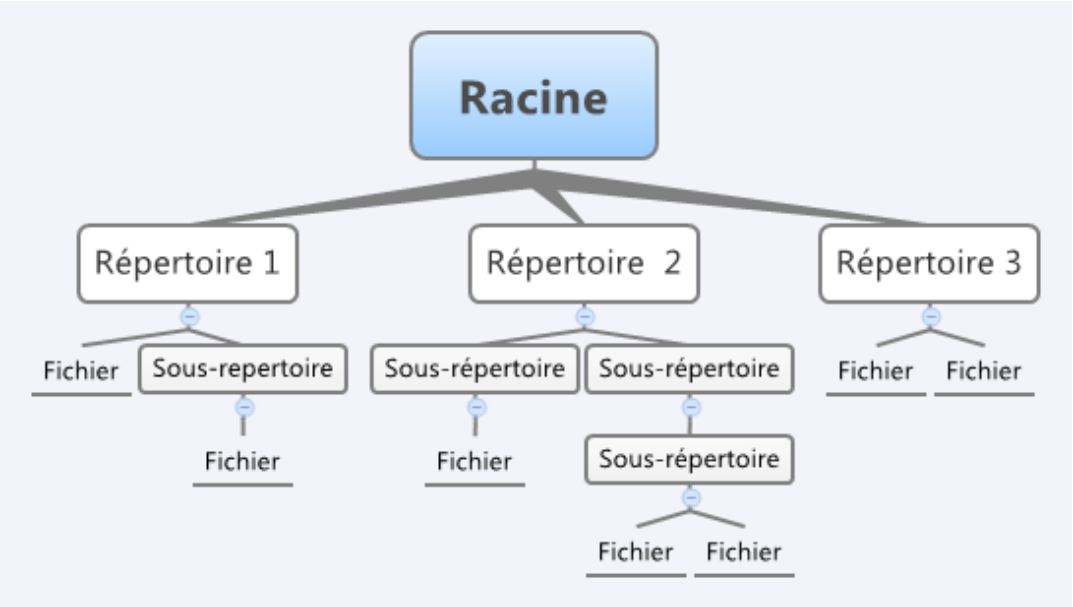
  — Données : $\approx 10^6$ fichiers sur mon PC (20 ans de données)

Du point de vue de l'utilisateur les fichiers sont organisés en une *structure arborescente* :

— Chaque fichier est contenu dans un répertoire.

— Ce répertoire, ainsi que d'autres répertoires ou fichiers, peuvent éventuellement être à leur tour contenus dans un autre répertoire.

— On peut ainsi remonter dans l'arborescence jusqu'à un répertoire appelée *répertoire racine*.



Arborescence des fichiers

On accède à une ressource (un fichier, par exemple) en donnant son adresse, appelée *chemin*. Il y a deux types de chemins associés à une ressource donnée.

— Le *chemin absolu*, donné à partir de la racine de l'arborescence.

— Les *chemins relatifs*, qui sont donnés à partir d'un point de l'arborescence. Il y a donc un chemin relatif par emplacement de l'arborescence : cet emplacement est le *répertoire courant*.

1.4.1 Sous Unix

— Arborescence unique : unifie l'accès aux différents disques (locaux, distants, disque dur, clé USB, etc.)

— Répertoire racine noté `/`

— $/a/b$: fichier ou sous-répertoire b du sous-répertoire a de la racine. Notation appelée *chemin absolu* du fichier ou répertoire b.

— a/b : fichier ou sous-répertoire b d'un sous-répertoire a (d'emplacement non précisé), c'est donc un chemin relatif.
Par exemple, si le répertoire courant est `/home/adent/` et qu'une commande fait référence à `Documents/trombi.pdf`, le fichier concerné se trouve au chemin absolu `/home/adent/Documents/trombi.pdf` !

1.4.2 Sous MS-Windows

— Chaque disque a un nom formé d'une lettre suivi de « : » (`A:`, ..., `Z:`), souvent source de problème (pas de convention standard).

— On utilise `\` au lieu de `/` pour noter les sous-répertoires.

1.4.3 Métadonnées

À chaque fichier/répertoire sont associées des informations, appelées *métadonnées* et notamment :

- taille du fichier
- propriétaire
- droits d'accès (lecture, écriture, exécution) pour le propriétaire, les membres de son groupe, les autres utilisateurs
- date de dernière modification, dernier accès

Utilisées par le système pour accorder/refuser l'accès, pour savoir ce qui doit être sauvegardé, etc.

1.5 Les programmes

Existent essentiellement sous deux formes :

Programmes en langage machine Ils contiennent le code des instructions du processeur qu'il faut mettre en mémoire pour exécuter le programme.

Scripts Programmes sous forme de texte dans un langage humainement compréhensible. Techniquement, le système va lancer un programme appelé *interpréteur* et lui donner ce texte. L'interpréteur va alors exécuter le programme.

1.5.1 Programmes en langage machine

- Difficiles à écrire directement, spécifiques à une architecture processeur/système d'exploitation donnée.
- En général produits par un *compilateur*, programme permettant de transformer un programme écrit dans un langage évolué en langage machine.

1.5.2 Scripts (sous Unix)

- Foultitude de langages disponibles (plus de 200 répertoriés sur wikipedia).
- Pour réaliser un script :
  - écrire le texte du programme dans un fichier texte
  - ajouter en première ligne : `#!` *chemin-vers-l'interpréteur* (par exemple `#! /bin/sh`)
  - Donner les droits d'exécution sur le fichier.

1.6 Protections

Avant de donner la main à un programme, le système d'exploitation bascule le processeur dans un mode appelé *mode utilisateur* (par opposition au *mode noyau*).

En mode utilisateur :

— On peut transmettre des demandes au système d'exploitation (*appels systèmes*).

— Il est impossible d'accéder aux périphériques directement.

— Les accès mémoires sont contrôlés et possibles uniquement dans une zone autorisée par le noyau. En cas de tentative d'écriture ou lecture ailleurs, le SE reprend la main et arrête le processus.

2 Environnement de développement intégré (IDE)

Logiciel permettant

— d'**écrire** des programmes

— de les **exécuter**

— de chercher l'origine d'un bogue (**débogueur**)

— de consulter la documentation du langage et des bibliothèques utilisées

La plupart des IDE sont spécialisés à un ou à quelques langages, d'autres sont plus généralistes et ouverts sur grand nombre de langages.

Dans ce cours :

— Langage : 

— IDE : IDLE (fourni avec Python)

Python :

— un des langages les plus utilisés dans le monde

— un des plus faciles à apprendre

2.1 Console interactive

On essaie successivement de taper :

```
2 + 3
a = 2
a
a=2
a       =     2
   a = 2
a + a
```

```
b + 1
b = 1
b + 1
b = b + 7
b = b + 5
b
2 ** 5
```

```
2 ** 1000                          "hello" + " world"
"hello"
"hello" + "world"
```

(démonstration interactive)

2.2 Éditeur

Ouverture d'une nouvelle fenêtre d'édition : menu File/New Window
On édite le texte suivant :

```
print("hello world!")
x = 42
```

Exécuter ce code : Run/Run Module (ou F5) (démonstration) Remarquez que python n'a pas écrit 42 :

Lorsqu'il s'exécute, un programme python n'affiche que ce qu'on lui demande explicitement d'afficher !

2.3 Débogueur

Dans l'éditeur on entre le programme suivant :

```
x = 231 / 77
z = 4
u = x**2 + 3 - z*3
t = z / u
print(t)
```

Exécution (F5) : erreur !

Que se passe t-il ?

On va regarder pas à pas en lançant le débogueur :

Menu Debug/Debugger (de la fenêtre d'interaction) : ouvre une fenêtre nommée Debug Control. Cocher la case « Source ».

Puis dans l'éditeur, exécuter le programme (F5). La fenêtre Debug Control montre la ligne qui va être exécutée et la valeur des variables avant cette exécution et la ligne en cours d'exécution est mise en surbrillance dans la fenêtre d'édition. Cliquer sur Step pour exécuter cette étape.