Operating Systems

September 24, 2019

1 Algorithmique et Pensée Computationnelle

2 Systèmes d'Exploitation

2.1 Logiciels système:

Logiciels intermédiaires avec lesquels les utilisateurs n'intéragissent pas directement mais qui restent nécessaires au bon fonctionnement de la machine (Windows, Linux, librairies, interprètes, etc.)

2.2 Applications:

Logiciels que l'utilisateur final utilise (Word, Chrome, jeux vidéos, etc.)

3 Systèmes d'Exploitation (Operating Systems)

L'OS est le logiciel le plus important de l'ordinateur car il est responsable de tous les autres programmes et de leurs priorités d'exécution, il organise aussi les interactions et l'orchestration des différents composants physiques de la machine.

L'OS contrôle l'accès au `matériel` physique et gère les `processeurs`, la `mémoire`, le `storage`, la `sécurité` et les `périphériques externes`. Par exemple, l'OS identifie quel utilisateur est connecté (au moment d'entrer le mot de passe à l'allumage), il reconnaît quelles touches du clavier et de la souris sont pressées, il affiche les images à l'écran et il sauvegarde les fichiers dans les `disques durs` internes ou externes.

La plupart du temps, de nombreuses applications sont exécutées au même moment sur le même ordinateur. Elles ont besoin d'utiliser les mêmes composants de l'ordinateur (`CPU`, `RAM`, `Storage`). Cependant, ces différents composants ne peuvent faire qu'une seule chose à la fois (même si ils les font très rapidement), c'est la raison pour laquelle l'OS orchestre chaque tâche de manière à ce que l'utilisateur ait l'impression que tout se déroule en même temps.

4 Interprètes et compilateurs

Les ordinateurs ne *"comprennent"* pas les langages de programmation, il faut passer par un programme qui va convertir le code écrit par un humain en instructions que l'ordinateur comprend (à

base de 0 et de 1).

Les interprètes et les compilateurs servent à faire ce travail de traduction, ils transforment donc les langages de programmation qui sont faits pour être compris et écrits par des humains, vers des instructions compréhensibles pour des ordinateurs.

La façon dont les interprètes et les compilateurs opèrent est différente et les deux approches apportent leur propre lot de bénéfices et d'inconvénients:

Les COMPILATEURS traduisent à l'avance

Les INTERPRÈTES traduisent au fur et à mesure

Programmes générés à l'avance (besoin d'anticiper tous les cas de figure)

Programmes générés au fur et à mesure

Le programme est intégralement traduit à l'avance

Les instructions sont traduites à la volée une par une

Le compilateur trouve les erreurs à la compilation

L'interprète relève les erreurs pendant l'exécution du programme

Plus rapide

Plus lent

Code machine généré et optimisé à l'avance

Code machine généré à la volée et optimisé à chaque exécution

Ex: C, C++, Java, Scala, Rust, etc.

Ex: Bash, Python, Javascript, etc.

5 Librairies

Une librairie est un ensemble de fonctions qui ont pour but d'être utilisées par d'autres programmes, mais une librairie seule ne suffit pas pour faire un programme. Par exemple, que ce soit `Google Chrome`, `Word`, ou `Instagram`, presque tous les programmes ont besoin d'utiliser des listes d'objets. Au lieu de réécrire l'ensemble des fonctions qui permettent de créer des listes et d'interragir avec, ces différents programmes utilisent tous une librairie écrite par un tiers.

Par exemple, Google Chrome utilise la librairie `std::vector` *qui sert à faire des listes plus de 20'000 fois:* Voir le code de Google Chrome

En `Python`, de nombreuses librairies sont disponibles d'emblée (`list`, `set`, `dict`, `str`, etc.) et de nombreuses autres peuvent être utilisées en utilisant la mot-clef `import`.

Dans l'exemple suivant, nous importons la librairie `math` qui propose de nombreuses fonctions mathématiques et nous essayons la fonction factorielle pour calculer `5!` .

[ ]:
```
import math

fact = math.factorial(5)
print(fact)
```