Ordinateur et environnement d'exécution

M319 — Programmation structurée

Jérôme Frossard

EPAI

8 mars 2026

Plan

Ordinateur (I/II)

Un ordinateur (*computer*) n'est pas une machine comme les autres. Sa fonction pratique dépend du **programme** qu'il exécute, et des **périphériques d'entrée/sortie** dont il est équipé. Ces périphériques permettent au programme d'**interagir avec le monde extérieur** :

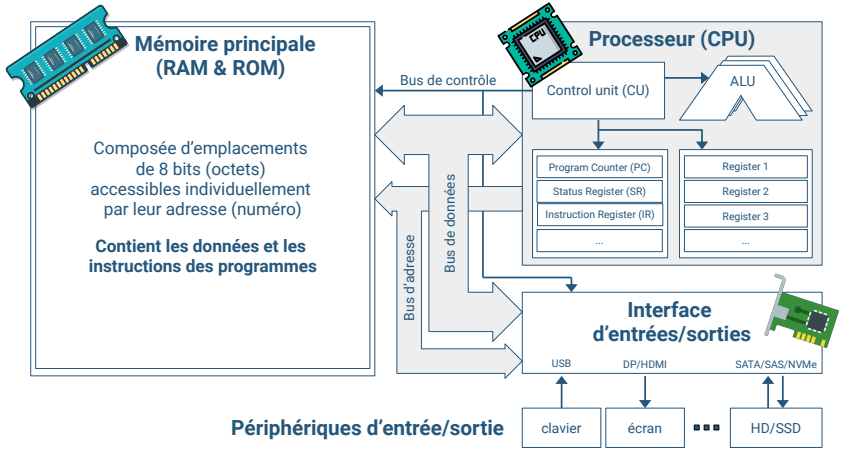
- terminal (ou émulateur de terminal)
- clavier, dispositif de pointage (souris, trackpad, etc.), carte graphique, carte audio, etc.
- barrette braille, clavier ergonomique, etc.

Un ordinateur est donc, dans une certaine mesure, une **machine universelle**. Avec le bon programme et les bons périphériques, il peut devenir n'importe quelle autre machine : une machine de traitement de texte, une borne d'arcade, un simulateur de conduite, un appareil photo, une machine à café.

Remarque : Il est important de noter que l'universalité d'un ordinateur concerne le calcul (la transformation des données). Les périphériques d'entrée/sortie sont spécifiques à une machine donnée ou à une catégorie de machines (ordinateurs de bureau, ordinateurs portables, etc.) et sont souvent eux-mêmes construits autour d'un petit ordinateur (système embarqué).

Ordinateur (II/II)

L'architecture de la plupart des ordinateurs modernes correspond, au moins au niveau logique, à l'**architecture de von Neumann** (ou architecture de Princeton).



Plan

Démarrage d'un ordinateur (*bootstrapping*)

Avec l'architecture de von Neumann, un programme doit se trouver dans la **mémoire principale** d'un ordinateur pour être exécuté. Il s'ensuit que le programme exécuté au démarrage doit déjà s'y trouver.

Ce premier programme ne peut pas être chargé depuis une mémoire secondaire, puisqu'il faudrait pour cela exécuter un programme. C'est le problème du ***bootstrapping*** : comment se soulever en tirant sur ses propres lacets ?

La solution est de charger ce programme à l'avance (souvent en usine) dans une partie de la mémoire principale constituée de **mémoire non volatile** (p. ex. une mémoire flash) :

- Dans un petit système embarqué (lave-vaisselle, machine à café, souris, clavier, etc.), ce programme est typiquement le programme qui contrôle la machine.
- Dans un PC ou un serveur d'entreprise, ce programme est le **firmware (BIOS/UEFI)** typiquement utilisé pour charger le **système d'exploitation (OS)**.

Plan

Environnement d'exécution

Pour un programme, le système d'exploitation est un **environnement d'exécution (*runtime*)** qui n'est pas l'ordinateur lui-même, mais une certaine abstraction de cette machine :

- Deux ordinateurs différents, mais compatibles (même architecture matérielle, p. ex. x86, amd64, arm64, etc.) avec le **même OS** apparaissent comme des machines similaires
- Deux ordinateurs strictement identiques avec des **OS différents** apparaissent comme des machines différentes.

Dans certains cas, l'environnement d'exécution est lui-même une abstraction de l'OS et de l'architecture matérielle. C'est le cas, par exemple, de la JVM (*Java Virtual Machine*), du .NET CLR, de Node.js ou encore de Python.

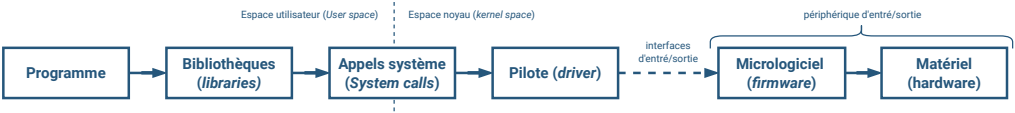
S'il existe une implémentation de l'environnement d'exécution pour leur **plateforme** (OS et architecture matérielle) respective, deux systèmes complètement différents peuvent apparaitre comme des machines similaires.

Remarque : Un navigateur, une suite bureautique, ou un shell sont aussi des environnements d'exécution, mais ce n'est pas leur fonction première. Un programme écrit pour un tel environnement est souvent appelé **script**.

Abstraction des périphériques

Avec un système d'exploitation, les périphériques d'entrée/sortie ne sont généralement accessibles que dans l'**espace noyau (*kernel space*)**, et c'est le plus souvent dans cet espace que fonctionnent les **pilotes de périphérique (*device drivers*)**.

Comme un programme d'application fonctionne dans l'**espace utilisateur (*user space*)**, il n'a pas accès aux périphériques eux-mêmes. Les opérations d'entrée/sortie commandent donc les périphériques de manière indirecte, à travers plusieurs couches d'abstraction logicielles (bibliothèques, appels système, pilotes, micrologiciels, etc.).



Dans un système d'exploitation, les **appels système (*system calls*)** constituent l'interface entre l'espace utilisateur, dans lequel fonctionnent les programmes d'application, et l'espace noyau, dans lequel fonctionnent les programmes qui gèrent le matériel.

Bibliothèques et API

Concrètement, un programme applicatif interagit avec le monde extérieur via des **API** plus ou moins standardisées, exposées par des **bibliothèques**. Par exemple :

- La bibliothèque standard d'un langage permet typiquement de lire et écrire dans les entrées/sorties standard (stdio) ou dans un fichier, et d'accéder au système de fichiers.
- Diverses API telles que celles des environnements de bureau, raw input, SDL, JavaFX, etc. permettent d'utiliser les périphériques HID (souris, trackpad, clavier, joystick, etc.).
- Les Berkeley sockets permettent de communiquer sur un réseau TCP/IP.

En développement d'applications, un programme n'est en principe pas conçu pour un matériel particulier, mais pour un environnement d'exécution. Toutefois, quel que soit l'environnement d'exécution, l'exécution d'un programme n'est possible que si les fonctionnalités des périphériques requis sont effectivement disponibles.

Remarque : Une API (*application programming interface*) est l'interface d'un composant logiciel (p. ex. une bibliothèque) qui décrit l'ensemble des opérations et des structures de données, fournies par ce composant.