Système d'exploitation (rappels)

M122 — Automatiser des procédures à l'aide de scripts

Jérôme Frossard

EPAI

24 novembre 2023

Plan

Plan

Qu'est-ce qu'un système informatique ?

Le mot « ordinateur » a été choisi en 1955 par le département de la publicité d'IBM France qui cherchait un terme plus vendeur que « ***electronic data-processing system*** » **(EDPS)**.

Ce terme est peut-être plus vendeur, mais il n'est certainement pas plus parlant. C'est pourquoi lui nous préférons une autre traduction : **système informatique**.

Un **système informatique** comprend :

- Un sous-système **matériel (*hardware*)**
- Un sous-système **logiciel (*software*)**

Le terme système informatique met en évidence l'**importance du logiciel**. Un ordinateur est une **machine universelle** qui n'a pas d'utilité propre. C'est en exécutant un programme qu'elle devient utile.

Exemples de système informatique

La première chose qui nous vient à l'esprit lorsque l'on parle de système informatique est bien sûr notre **ordinateur personnel** (PC).

On peut également penser à des appareils telles que :

- Les **téléphones portables (*smartphone*)** iOS et Android
- Les **serveurs** d'entreprise ou des centres de données

Mais il en existe encore beaucoup d'autres. En effet, on trouve des systèmes informatiques dans presque tous les appareils du quotidien, du lave-vaisselle à la télévision en passant par la machine à café, le routeur wifi et l'appareil photo. Ce sont les **systèmes embarqués**.

Sous-système matériel

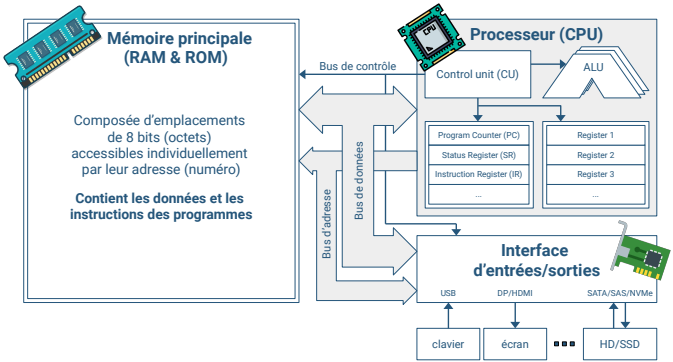
Le **sous-système matériel (*hardware*)** est typiquement constitué :

- D'une alimentation électrique très stable
- D'un **CPU**, d'une **mémoire principale** et d'**interfaces d'entrées/sorties** sur une carte mère (*motherboard*) et des cartes d'extensions (*expansion cards*)
- De **périphériques d'entrées/sorties** :
  - Une ou plusieurs unités de **mémoire secondaire** (disques durs, SSD, etc.)
  - Un ou plusieurs **écrans** connectés à une carte graphique
  - Un **clavier** et un ou plusieurs **dispositifs de pointage** (souris, trackpad, écran tactile, etc.)
  - Un micro et des hautparleurs connectés à une carte son
  - Une ou plusieurs **cartes réseau** (Ethernet, WiFi, etc.)

Sous-système matériel — Architecture de Princeton (I/IV)

La manière dont la partie matérielle de nos systèmes informatiques est conçue correspond généralement à l'**architecture de Princeton** (ou de von Neumann). Dans cette architecture, la mémoire principale contient à la fois les données et les instructions des programmes. Par contraste, dans l'architecture de Harvard, données et instruction sont stockées dans des mémoires séparées.

Cette architecture a été décrite pour la première fois dans un rapport écrit à l'IAS de **Princeton** en **1945,** par **John von Neumann,** à propos de l'architecture de l'**EDVAC**.



Sous-système matériel — Architecture de Princeton (II/IV)

Le **CPU ou unité centrale de traitement** (processeur) se compose d'un ou plusieurs cœurs. Chaque cœur comprend :

- Plusieurs **registres** qui sont des mémoires très rapides d'une capacité de 1 mot*.
- Un ou plusieurs **unités arithmétiques et logiques (ALU)** pour effectuer des opérations.
- Une **unité de contrôle (CU)** pour orchestrer les différents composants.

La **mémoire principale** est rapide, directement accessible par le CPU et contient à la fois les données à traiter et les instructions des programmes en cours d'exécution.

Les **interfaces d'entrées/sorties** permettent l'échange de données avec les **périphériques d'entrées/sorties** (écran, clavier, disques durs, SSD, etc.).

***** Un mot est la longueur maximale, exprimée en bits, d'un nombre que le CPU peut traiter en une seule opération. Dans nos ordinateurs, la longueur d'un mot est généralement de 64 bits (8 octets).

Sous-système matériel — Architecture de Princeton (III/IV)

Les composants sont interconnectés par l'intermédiaire de **bus**. Un bus est un système de transmission de données où tous les composants utilisent les mêmes lignes. Ce qui permet de réduire le nombre de lignes nécessaires par rapport à des liaisons point à point.

Dans l'architecture de Princeton, les bus principaux sont :

- Le **bus d'adresse** permet de sélectionner un emplacement de la mémoire principale ou un registre d'un périphérique.
- Le **bus de données** permet de lire ou d'écrire une donnée dans l'emplacement de la mémoire principale ou le registre de périphérique sélectionné.
- Le **bus de contrôle** permet d'activer ou désactiver les différents composants connectés au bus et de définir le sens des transferts de données.

Les bus d'adresse et de données sont constitués de **lignes parallèles**. Dans nos PC, il en a typiquement 64 pour le bus de données et entre 36 et 40 pour le bus d'adresse.

Sous-système matériel — Architecture de Princeton (IV/IV)

Le CPU a un **accès direct uniquement** aux données et instructions qui se trouvent dans la **mémoire principale**. Il n'a pas d'accès direct aux données des périphériques.

Pour échanger des données avec un périphérique, le CPU **exécute un programme** appelé **pilote de périphérique** (*device driver*) qui lui permet d'écrire une séquence de commandes dans un registre de contrôle du périphérique.

En réponse, le périphérique effectue une tâche qui peut être, par exemple :

- Écrire une donnée dans un registre de statut
- Démarrer un transfert de données dans ou depuis la mémoire principale

Lorsque la tâche est terminée, le périphérique le **signale** souvent à l'aide d'une **interruption**.

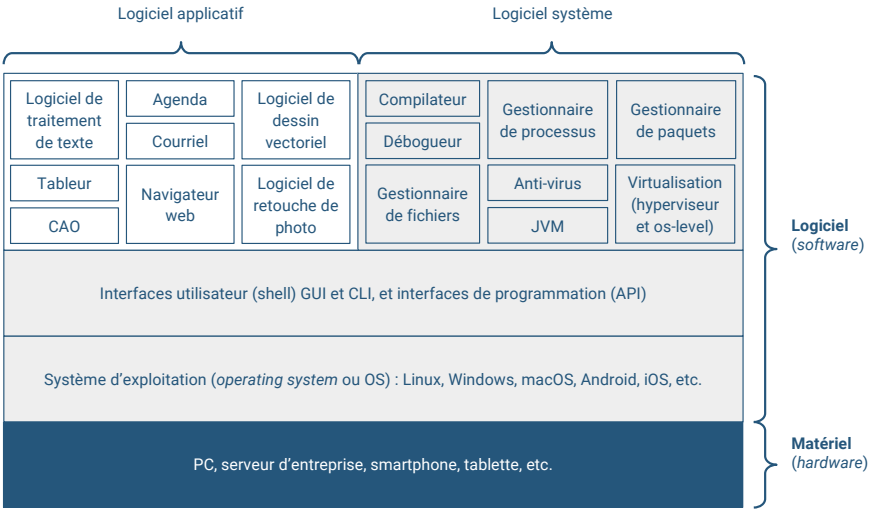
Remarque : Un périphérique est aussi un système informatique. Son logiciel, appelé ***firmware***, en détermine la fonction, et n'est modifié que pour la correction d'erreur ou un éventuel ajout de fonctionnalités.

Sous-système logiciel

Le **sous-système logiciel (*software*)** est constitué de l'ensemble des programmes. Ceux-ci peuvent être classés en deux catégories :

- Le **logiciel système** : ensemble des programmes qui participent au fonctionnement du système informatique, ou qui permettent de réaliser des tâches liées à son l'administration. Cela correspond pour l'essentiel aux systèmes d'exploitation (Linux, macOS, Windows, etc.) sur lesquels nous revenons plus loin.
- Le **logiciel applicatif** : ensemble des programmes qui permettent de réaliser des tâches avec le système informatique, par exemple :
  - Suite bureautique (LibreOffice, Microsoft 365, etc.)
  - Client de messagerie (Gmail, Outlook, etc.), navigateur web (Firefox, Chrome, etc.)
  - IDE (Visual Studio Code, Netbeans, etc.) et contrôle de version (git, mercurial, etc.)
  - Forge logicielle (GitLab, GitHub, etc.)
  - Progiciel : CRM, ERP, PIM, PLM, SCM, etc.

Sous-système logiciel — Vue d'ensemble



Plan

Démarrage d'un système informatique

Nous savons que :

- La mémoire principale est volatile (elle a besoin d'énergie pour conserver les données).
- Au démarrage du système, la mémoire principale est donc vide.
- Pour charger un programme dans la mémoire depuis une mémoire secondaire (p. ex. un disque dur), le CPU doit exécuter un programme.

Mais :

- Pour être exécuté, un programme doit se trouver dans la mémoire principale.

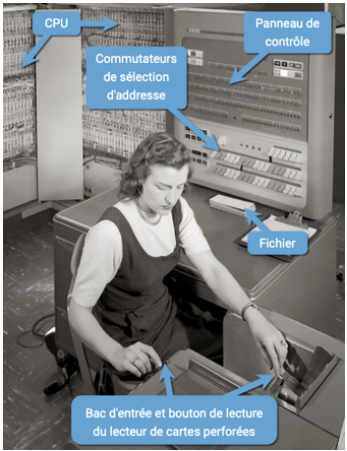
La question qui se pose est :

- Comment fait-on pour charger ce premier programme dans la mémoire principale ?

Démarrage d'un système informatique — Démarrage manuel (I/II)

Jusqu'au tournant des années 1970, les ordinateurs sont de gros ordinateurs centraux (*mainframes*) dont le démarrage consiste à :

- Charger le support du programme (carte perforée, bande magnétique, ou ruban perforé), dans le lecteur approprié.
- Mettre le lecteur en attente en actionnant un commutateur sur le lecteur lui-même.
- Sélectionner l'adresse à laquelle charger le premier programme et l'adresse du lecteur sur le panneau de contrôle.
- Démarrer le transfert du lecteur vers la mémoire principale en actionnant un commutateur sur le panneau de contrôle.



Démarrage d'un système informatique — Démarrage manuel (II/II)

Dans les années 1970, les circuits intégrés (IC) permettent l'apparition de nouveaux types de systèmes, plus petits et moins chers. D'abord les mini-ordinateurs (mini) puis, avec l'arrivée des microprocesseurs, les micro-ordinateurs (micro).

Le démarrage des minis (à droite) et des premiers micros (en bas) est similaire à celui des mainframes. Il est même un peu plus compliqué, car le programme de pilotage du périphérique qui était câblé doit maintenant être entré à la main instruction par instruction avec les commutateurs du panneau de contrôle.

Démarrage d'un système informatique — ROM et BASIC

Très rapidement, le démarrage des micro-ordinateurs est simplifié par l'**ajout d'une petite quantité de mémoire morte (ROM)** dans la mémoire principale. Cette mémoire, qui n'est accessible qu'en lecture, est chargée en usine avec le programme que le système doit exécuter au démarrage.

Au tournant des années 1980, tous les micros ont un **clavier** et une **sotie vidéo analogique**. Presque tous ont une ROM avec un **interpréteur BASIC**. Celui-ci permet de programmer, mais aussi de charger ou d'enregistrer données et programmes depuis ou dans une mémoire secondaire, typiquement une cassette ou une **disquette** (*floppy disk*).

Démarrage d'un système informatique — IBM PC et BIOS

Au début des années 1980, IBM lance l'**IBM PC**, un micro pour le marché professionnel.

Les utilisateur·rice·s de cet ordinateur doivent être en mesure de l'utiliser sans avoir à apprendre à programmer. Pour cela, il doit être équipé d'un système qui permet de l'exploiter facilement. Il doit être équipé d'un **système d'exploitation**.

Comme IBM a préféré sous-traiter le développement du système d'exploitation, ce dernier ne se trouve pas dans la ROM. À la place, elle contient les composants logiciels suivants :

- Un programme de démarrage capable de charger un programme depuis un lecteur de disquette (plus tarde depuis un disque dur) et d'en lancer l'exécution.
- Un ensemble de sous-programmes qui facilite le pilotage des périphériques : le **Basic Input/Output System ou BIOS**.

Le logiciel de la ROM est appelé **firmware**.

Plan

Un peu de contexte — Mécanographie (I/II)

Les premiers ordinateurs arrivent dans les administrations publiques et les entreprises au tournant des années 1950, mais ils ne constituent pas d'emblée une révolution. Ils s'intègrent parfaitement dans des systèmes de traitement de données déjà très mature (plus de 50 ans), construits autour des **ateliers de mécanographie** et de leurs machines électromécaniques : **tabulatrices**, trieuses, interclasseuses, imprimantes, etc.

Les données sont saisies sous forme numérique sur des **cartes perforées** à l'aide de perforatrices par un très grand nombre d'opératrices.

Ces données proviennent de listes de transactions bancaires, de fiches de paye, de bulletins de commandes, de données statistiques, etc.

Un peu de contexte — Mécanographie (I/II)

Dans les ateliers de mécanographie, les ordinateurs remplacent peu à peu les tabulatrices.

Celles-ci peuvent réaliser directement des additions, mais la réalisation de multiplication ou de division nécessite plusieurs passages dans la machine, ce qui rend ce genre d'opération, lent et compliqué.

Les ordinateurs peuvent réaliser des calculs beaucoup plus complexes, plus rapidement et en un seul passage. Mais la manière de travailler reste essentiellement la même :

- Les données sont entrées dans la machine avec le même type de lecteur de cartes perforées que celui des tabulatrices.
- Les résultats sont imprimés sur les mêmes imprimantes, ou stockés sur des cartes perforées avec les mêmes perforatrices.

De fait, le **traitement par lots** mis en place et perfectionnés depuis les années 1910 reste donc applicable.

Traitement par lot et multiprogrammation

Contrairement à la majorité de ceux que nous utilisons aujourd'hui, les premiers ordinateurs ne sont pas interactifs ; les données sont traitées par lots.

Le **traitement par lots** (*batch processing*) peut être décrit de la manière suivante :

- Les données sont saisies sur des cartes perforées par un grand nombre d'opératrices.
- À intervalle régulier, les cartes sont rassemblées en paquets et vérifiées.
- Les paquets de cartes sont convoyés vers l'atelier de mécanographie.
- Les paquets de cartes qui contiennent des **données devant être traitées ensemble** sont regroupés pour former des **lots** de données.
- Les lots sont placés dans une **file d'attente**.
- L'**ordre de traitement** des lots est déterminé par le chef d'atelier en fonction de facteurs tels que l'importance et l'urgence de la tâche, la disponibilité des machines, etc.
- Chaque opération du traitement est appliquée à l'ensemble des cartes du lot.
- Les résultats sont imprimés ou stockés sur de nouvelles cartes perforées.

Bande magnétique et moniteur (I/III)

Dans le courant des années 1950, avec l'augmentation de la puissance de calcul et de leur capacité de stockage, les ordinateurs finissent par remplacer toutes les machines, en particulier les trieuses et les interclasseuses. Mais ils sont aussi beaucoup plus chers.

Dans ces conditions, l'introduction manuelle des lots de cartes, l'impression, et le stockage sur cartes perforées des résultats sont beaucoup trop lents et ne permettent pas un taux d'utilisation suffisant.

Le processus est donc adapté pour tirer parti des nouvelles **unités de stockage à bande magnétique**, qui ont une densité de stockage beaucoup plus grande, et qui sont **bien plus rapides**.

Bande magnétique et moniteur (II/III)

L'ordinateur est connecté uniquement à des unités de stockage à bande magnétique :

- Une pour les lots de données
- Une pour les résultats
- Une pour les programmes

Les données sont saisies, vérifiées, convoyées et groupées en lots comme avant, mais au lieu de charger les lots dans l'ordinateur un à un, ils sont copiés **hors-ligne** (sans utiliser l'ordinateur) sur des bandes magnétiques.

Sur l'ordinateur, un programme appelé **moniteur** lit l'entête d'un lot sur la bande magnétique, charge le programme pour le traitement de ce lot et lui passe le contrôle. Le programme traite les données et enregistre les résultats sur la bande magnétique, puis rend le contrôle au moniteur qui passe au lot suivant.

Les données de la bande magnétique des résultats sont imprimées ou stockées sur des cartes perforées, là encore, **hors-ligne**.

Bande magnétique et moniteur (III/III)

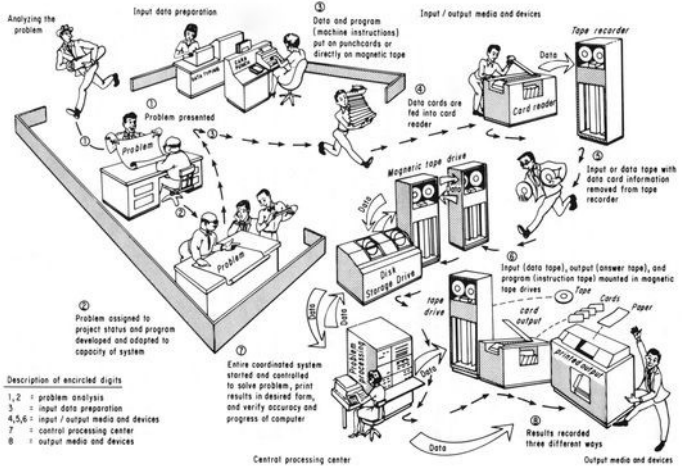


Fig. 1-1. A typical business application of an electronic data-processing system

Multiprogrammation

Dans les années 1960, la vitesse de traitement des ordinateurs augmente encore. La durée d'un cycle du processeur est de l'ordre de la centaine de microsecondes.

Comme les temps d'accès aux périphériques d'entrées/sorties sont de l'ordre de la seconde, le CPU doit attendre plusieurs milliers de cycles sur les périphériques, temps durant lequel il pourrait exécuter des centaines voire des milliers d'opérations.

Une solution est de modifier le moniteur pour qu'il puisse charger plusieurs programmes en même temps. Ainsi, si un programme doit attendre sur un périphérique, le moniteur passe le contrôle à un autre programme. C'est la **multiprogrammation**.

Un risque lié à la multiprogrammation est la modification des données ou des instructions d'un programme par un autre programme. La mitigation de ce risque a conduit aux mécanismes d'isolation des **processus** de nos systèmes d'exploitation. Nous y reviendrons.

Plan

Interaction avec l'ordinateur

Avec le **traitement par lots**, il n'y a **aucune interaction entre l'utilisateur·rice et le programme** durant le traitement du lot de données. C'est encore le cas, par exemple, pour :

- Le traitement des virements interbancaire durant la nuit.
- L'exécution d'une tâche planifiée.

Le traitement par lots permet d'avoir un fort taux d'occupation du CPU, mais attendre une journée pour avoir le résultat de la compilation d'un programme n'est pas idéal.

C'est pourquoi les ingénieurs ont rapidement cherché un moyen efficace pour envoyer des commandes à un programme et recevoir immédiatement une réponse. En d'autres termes, un moyen efficace de réaliser un **programme interactif**.

La partie système informatique qui permet à un·e utilisateur·rice d'interagir avec un programme est appelée **interface utilisateur·rice**. Elle se compose de matériel et de logiciel.

Téléscripteur

Dans la deuxième moitié des années 1940, les réseaux Telex commencent à remplacer le télégraphe. À la place de manipulateur pour le code morse, le réseau Telex utilise des machines à écrire électromécaniques appelées **téléscripteur (*teletype* ou TTY)**.

Lorsqu'une touche du téléscripteur est pressée, celui envoie le code du caractère sur **ligne sérielle** à un téléscripteur distant. Ce dernier imprime le caractère et renvoie le même code au premier. Le renvoi du caractère (**écho**) permet à l'utilisateur·rice de voir les caractères envoyés.

En remplaçant le deuxième téléscripteur par l'ordinateur, un·e utilisateur·rice peut envoyer des caractères à un programme, et recevoir sur le papier les caractères que ce dernier envoie en retour.

Temps partagé (I/II)

Dans les années 1960, la vitesse des ordinateurs permet d'envisager d'y connecter plusieurs téléscripteurs pour permettre à plusieurs personnes de l'utiliser en même temps.

Une solution est de modifier le moniteur pour charger plusieurs programmes et pour passer d'un téléscripteur à l'autre à intervalle régulier. Pour chaque téléscripteur, le moniteur a une structure de données appelée **session** qui contient notamment l'état du programme avec lequel l'utilisateur·rice interagit.

Lorsque le tour d'une session arrive, le moniteur charge les données et passe le contrôle au programme correspondant durant une courte période (*time slice*) de l'ordre de 10 à 100 millisecondes. À la fin de la période, une interruption rend le contrôle au moniteur, qui enregistre l'état de la session et passe à la suivante.

L'ordinateur doit être équipé d'un timer qui lève une interruption à intervalle régulier.

Temps partagé (II/II)

Il y a un grand nombre d'exemples de système à temps partagé. On peut par exemple citer :

- Le *Compatible Time-Sharing System* (CTSS) développé au MIT au début des années 1960 et précurseur de Multics. C'est le premier système à temps partagé.
- Multics (MULTiplexed Information and Computing System) dont le développement démarre 1964 et qui constitue une étape clé dans le développement des systèmes d'exploitation. Il a également été une des principales sources d'inspiration pour le développement UNIX à la fin des années 1960.
- Le *Dartmouth Time-sharing Tystem* (DTSS) développé au Dartmouth College pour permettre aux étudiants de premier cycle de se familiariser avec le BASIC.

Avec la multiprogrammation, le temps partagé a conduit au système d'isolation des processus et au multitâche préemptif de nos systèmes d'exploitation.

Interpréteur de commande et CLI

Un programme avec lequel l'utilisateur·rice interagit en envoyant des commandes sous forme de texte avec un téléscripteur est un **interpréteur de commande**.

Une interface utilisateur·rice composée d'un téléscripteur et d'un interpréteur de commande est une **interface en ligne de commande (*command line interface* ou CLI)**

L'interpréteur de commande envoie une **invite de commande (*prompt*)** sur le téléscripteur pour informer qu'il est prêt à recevoir une commande. Ce principe qui semble évident a été mis en œuvre pour la première fois dans le CTSS.

Aujourd'hui, les **interpréteurs de commande des systèmes d'exploitation (shell)** ont tous une invite de commande similaire à celle du CTSS.

Terminal

Un téléscripteur connecté à un ordinateur est appelé « terminal d'ordinateur » ou juste « **terminal** ». Avec l'arrivée des circuits intégrés (IC) et des microprocesseurs, il est devenu possible de réaliser des terminaux avec un écran à la place du papier.

Dans les années 1980, les terminaux étaient souvent construits autour des mêmes microprocesseurs que les micro-ordinateurs. Les terminaux haut de gamme avaient même une puissance de calcul supérieur à celle de la plupart des micros.

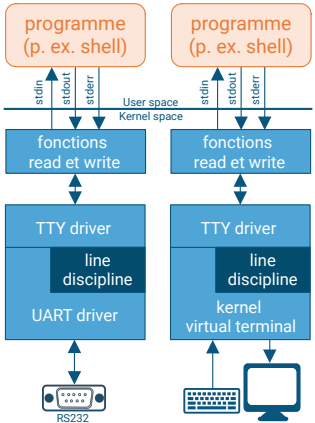
L'illustration montre un terminal **vt100** de Digital Equipment Corporation (DEC). Ses fonctionnalités constituent aujourd'hui encore un standard.

Terminal sous Linux

Linux est un système d'exploitation proche des systèmes Unix. Par défaut, l'interface du système est une interface en ligne de commande avec un shell similaire à celui d'Unix.

Comme nos ordinateurs ont déjà un clavier et un écran, un terminal est inutile. Le noyau (*kernel*) du système comprend un sous-programme qui simule un terminal en utilisant le clavier et l'écran. Pour les programmes, ce terminal est identique à un terminal connecté à une interface RS232.

Le sous-programme « line discipline » s'occupe de l'**écho**, du traitement des caractères spéciaux (backspace, delete, etc.), et de l'envoi des caractères au programme lorsqu'il reçoit un caractère *line feed* (LF). Le programme communique avec le terminal en utilisant les **entrées/sorties standards** (stdin, stdout et stderr) comme des fichiers.

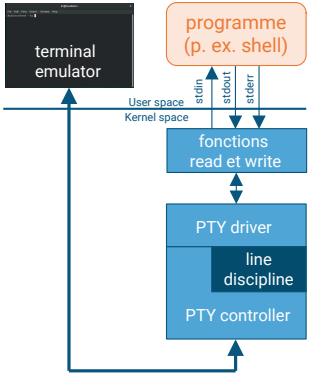


Pseudo-terminal et émulateur de terminal

Sous Linux avec une interface graphique ou sous Windows, le système utilise l'écran pour dessiner les fenêtres et envoie les caractères du clavier à la fenêtre active.

Dans ce cas, on utilise un programme appelé **émulateur de terminal** qui représente l'écran du terminal dans une fenêtre. Comme ce programme ne se trouve pas dans le noyau, il ne peut pas communiquer directement avec le *TTY driver*. Pour cela, le système peut créer un terminal qui n'est lié à aucun matériel : un **pseudo-terminal (PTY)**.

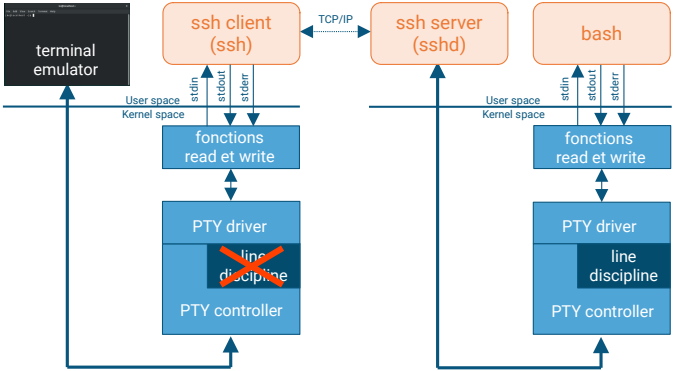
Pour le programme, cela ne fait, encore une fois, aucune différence qu'il s'agisse d'un terminal matériel, du terminal virtuel du noyau, ou d'un émulateur de terminal.



Pseudo-terminal et SSH

Le Secure Shell (**ssh**) permet de connecter un émulateur de terminal à un programme distant, généralement l'interpréteur de commande du système d'exploitation, par exemple, bash sous Linux, et cmd ou powershell sous Windows.

La machine cliente exécute le **client ssh** (ssh). La machine distante exécute le **serveur ssh** (*ssh daemon* ou sshd). La configuration du serveur ssh spécifie le programme à lancer à l'ouverture d'une session.



Terminal et entrées/sorties standard

Un programme avec une interface en ligne de commande (CLI), comme ceux que vous avez réalisés en Java, ne communique pas directement avec le pilote du terminal, mais utilise des **entrées/sorties standards (*stdio*)**.

Pour le programmeur, les *stdio*s sont **comme des fichiers**.

- L'**entrée standard** (*stdin*) est un fichier en lecture seule (*read only*) et, par défaut, les caractères viennent du clavier du terminal.
- La **sortie standard** (*stdout*) et la **sortie d'erreur** (*stderr*) sont des fichiers en écriture seule (*write only*) et, par défaut, les caractères sont affichés sur l'écran du terminal.

L'utilisation des *stdio*, plutôt que le pilote du terminal, permet de remplacer le terminal par d'autres « trucs » qui se comportent comme des fichiers (un « vrai » fichier, une *stdio* d'un autre processus, un canal TCP, etc.) sans modifier le programme. C'est ce que l'on appelle **redirection d'entrées/sorties**.

Remarque : En Java, `System.in` correspond à *stdin*, `System.out` à *stdout* et `System.err` à *stderr*.

Plan

Qu'est-ce qu'une unité de stockage

Une unité de stockage est un périphérique d'entrée-sortie qui permet de stocker de l'information sur un support de données non volatile, le plus souvent magnétique (disque dur, bande magnétique) ou à semi-conducteur (mémoires flash NOR ou NAND).

Comme la taille d'une unité de stockage est généralement grande par rapport à la mémoire principale, on parle souvent d'**unité de stockage de masse**.

Les unités de stockage sont des **périphériques de type bloc (*block device*)**, par contraste avec les périphériques de type octet (*byte device*) comme un clavier par exemple.

- Dans un périphérique de type bloc, les données sont organisées en bloc et ces blocs sont accessibles dans n'importe quel ordre. Leur taille varie d'un périphérique à l'autre, mais est typiquement de 512 octets, 4 ko ou 64 ko.
- Dans un périphérique de type octet, les données sont accessibles uniquement de manière séquentielle et il n'est souvent pas possible de revenir en arrière.

Mesurer la taille des unités de stockage

De manière générale, depuis la fin des années 1960, l'**octet (ou *byte*)** est l'unité de mesure de la **quantité de mémoire**. Il est important de ne pas confondre le **byte** avec le **bit** (*binary digit*) qui est l'unité de mesure de la **quantité d'information**.

Pour représenter de grandes quantités de mémoire, on peut utiliser des multiples de l'unité. Ceux-ci peuvent être décimaux (SI) ou binaires (CEI) :

Préfix décimal	Unité	Quantité [octet]
kilo-	ko, kB	10^3
méga-	Mo, MB	10^6
giga-	Go, GB	10^9
téra-	To, TB	10^{12}

Préfix binaire	Unité	Quantité [octet]
kibi-	KiB	2^{10}
mébi-	MiB	2^{20}
gibi-	GiB	2^{30}
tébi-	TiB	2^{40}

À l'usage, il a beaucoup d'incohérences. Par exemple, 1GB peut correspondre au multiple décimal, binaire ou à un mélange des deux (p. ex. 1'000'000 x 1024 octets).

Qu'est-ce qu'un fichier ?

Un fichier peut être défini comme une séquence d'octets, d'une longueur définie, avec un nom, et codé sur un support physique.

Un fichier peut être, par exemple :

- Une séquence d'enregistrement sur une bande magnétique
- Une liste de blocs sur un disque dur ou un SSD.

On peut distinguer différents types de fichiers :

- Fichier de texte : contient uniquement des codes de caractères, par exemple, un fichier source (.java ou .py) ou un fichier de configuration (.ini, .cnf, etc.).
- Fichier séquentiel : contient une séquence d'enregistrements de même structure ; leurs cas d'utilisation sont souvent ceux d'un SGBD embarqué comme SQLite.
- Fichier binaire : tout le reste, par exemple, une image (.jpg, .pgn, .gif, etc.), une vidéo (.mov, .mpg, etc.) ou un exécutable (.exe, .dll).

Organisation des fichiers

Lorsque le nombre de fichiers devient grand, il est important de les organiser.

Une organisation hiérarchique est généralement assez efficace. Dans le monde physique, ce mode d'organisation peut se traduire de la manière suivante :

- Un bâtiment contient des pièces,
- Une pièce contient des armoires,
- Une armoire contient des tiroirs,
- Un tiroir contient des fichiers.

Sur un support de données, ce mode d'organisation se traduit de la manière suivante :

- La racine contient des **répertoires (*directory*)** et des fichiers
- Un répertoire contient des répertoires (sous-répertoires) et des fichiers.

Remarque : Les répertoires sont aussi appelés dossiers (*folder*)

Organisation des unités de stockage

Une unité de stockage comme un disque dur ou un SSD est composée de cellules appelées **bloc** de 512 octets, 4 ko ou même 64 ko. Avec le *logical block addressing* (LBA), chaque bloc a une adresse de 48 bits qui permet d'y accéder.

On peut distinguer trois ou quatre niveaux d'organisation :

- Une unité de stockage ou un RAID (*Redundant Array of Independent Disks*) est un **volume physique**.
- Un volume physique est divisé en **partitions**.
- Les partitions peuvent être regroupées ; et les groupes, divisés en **volumes logiques** (p. ex. LVM)
- Un système de fichiers est installé sur une partition ou un volume logiques

Un **système de fichiers** permet une **organisation hiérarchique** des fichiers. Le **formatage** est la création d'un système de fichier dans une partition. Il ne peut y avoir qu'un système de fichier par partition.

Système de fichiers

Parmi les principaux systèmes de fichier, on peut mentionner :

- **Windows : NTFS**, FAT32, FAT, etc.
- **Linux : ext4**, ext3, Btrfs, etc.
- **macOS : APFS**, HFS+, etc.

Les principales fonctionnalités d'un système de fichier sont :

- Gestion d'un catalogue du contenu du disque
- Gestion de métadonnées pour les fichiers et les répertoires
  - Nom et taille
  - Date de création, d'accès, de modification
  - Propriétaire, permissions (read, write, execute, etc.), ACL (*access control list*)
- Gestion de l'espace (allocation et libération des blocs)
- Journalisation des opérations (pour le système de fichiers journalisé, notamment ext4, NTFS et APFS)

Plan

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

Un **système d'exploitation** (*operating system* ou OS) est un système logiciel composé d'un ensemble de programmes qui a pour but de faciliter l'exploitation du système informatique par les utilisateur·rice·s, et de faciliter son administration.

On peut distinguer au moins cinq grandes familles :

- Les systèmes Windows : Windows 10, Windows 11, Windows Server, etc.
- Les systèmes d'Apple : OSX, macOS
- Les systèmes GNU/Linux : Debian, Ubuntu, Fedora, RHEL, etc.
- Les systèmes mobiles : iOS, Android
- Les systèmes pour l'embarqué : TinyOS, LiteOS, FreeRTOS, QNX, etc.

Il est important de noter qu'on utilise le **système informatique présenté par l'OS**, par la machine physique sur lequel il fonctionne. On utilise Windows, Linux ou macOS, pas le modèle d'ordinateur XYZ.

Noyau et interfaces

Un système d'exploitation est généralement composé de deux parties :

- L'**espace utilisateur** (*user space*)
- L'**espace noyau** (*kernel space*)

Les composants de l'« espace utilisateur » sont les programmes qui peuvent être exécutés par un·e utilisateur·rice ou un·e administateur·rice.

Le noyau comprend tous les composants logiciels qui ne sont pas directement accessibles. Ces composants exécutés par le CPU en **mode protégé** pour éviter les risques de corruption par des programmes de l'espace utilisateur.

Le logiciel du noyau a directement accès à la mémoire et au matériel. Le logiciel de l'espace utilisateur n'a accès qu'à son propre **espace de mémoire virtuelle**, et n'accède au matériel que par l'intermédiaire des **pilotes de périphériques**.

Fonctionnalités du noyau

Gestion des processus : création, exécution, terminaison, surveillance, communication interprocessus (IPC), ordonnancement, etc.

Gestion de la mémoire : allocation et libération d'espace dans la mémoire virtuelle, *swapping*, etc.

Gestion des périphériques : pilotes de périphériques (*device drivers*), clavier, dispositifs de pointage, carte graphique, unités de stockage (détection, création de partitions, formatage, etc.), interfaces réseau (configuration des protocoles IP, DHCP, DNS, etc.).

Gestion des fichiers : création, suppression, manipulation de fichiers et de répertoire, gestion des permissions, etc.

Gestion de la sécurité : contrôle d'accès (authentification et autorisation), chiffrement des unités de stockage, pare-feu, etc.

Journalisation d'événements : traçabilité, surveillance, dépannage, etc.

Interfaces

Les fonctionnalités du noyau sont accessibles à travers deux types d'interfaces :

- Des interfaces utilisateur·rice : **shell**
- Des interfaces de programmation : API ou *application programming interface*

Les API sont destinées aux programmeur·euse·s et permettent de réaliser des applications pour le système d'exploitation, comme la **Win32 API** sous Windows ou la **C Standard library** sous Linux.

Les shells sont destinés aux utilisateur·rice·s (*user*) et aux adminitrateur·rice·s (*super user*, *root*, *administrator*, etc.). On en distingue deux sortes :

- Les shells graphiques (GUI) : Windows, macOS
- Les shells en ligne de commande (CLI) : **bash**, **powershell**, etc.

Dans ce module nous ne nous intéressons qu'aux shells en ligne de commande.

Shell

Pour l'utilisateur·rice, un shell GUI ou CLI a deux fonctions principales :

- Gérer les fichiers
- Lancer l'exécution de programmes (applications ou utilitaires système)

Avec un shell GUI, on utilise l'**explorateur de fichier** (directement ou depuis une application) pour la première, et on **tape** ou on **double clique** sur une **icône** pour la seconde.

Avec un shell CLI, on ouvre une fenêtre de terminal connecté à un shell (p. ex. **bash** ou zsh), puis :

- On utilise les commandes de gestion (**ls**, **cd**, **pwd**, **file**, **cat**, **etc**.) du système de fichier.
- On tape le **chemin complet** du fichier exécutable en spécifiant d'éventuelles **options**.

Anatomie d'une commande

Dans un shell, une commande est généralement écrite de la manière suivante :

- Le nom de la commande
- Zéro, une ou plusieurs options qui commence par **-** ou **--**
- Un ou plusieurs arguments, généralement les chemins des objets sur lesquels on applique la commande

Par exemple, la commande suivante liste le contenu du répertoire courant :

```
ls -a -l .
```

Dans cette commande :

- **ls** est le nom de la commande
- **-a** et **-l** sont des options
- **.** est l'argument (le . représente le répertoire courant).

Gestion de fichier dans le shell

Quand on utilise un shell (dans une fenêtre de terminal), on se trouve dans un répertoire. Ce répertoire, le répertoire courant, est appelé **répertoire de travail (*working directory*)**. Par défaut, il s'agit du **répertoire personnel (*home directory*)** de l'utilisateur·rice.

Pour gérer nos fichiers, on a d'abord besoin de commandes qui permettent de se déplacer dans la hiérarchie de répertoires et d'afficher leur contenu. En voici quelques-unes :

- **pwd** : affiche le chemin du répertoire de travail (*print working directory*)
- **ls** : liste le contenu du répertoire.
- **ls -al** : avec les **options a** et **l** la commande affiche aussi les fichiers cachés.
- **cd CHEMIN'** : aller dans le répertoire spécifié dans l'**argument** (*change directory*).
- **cd ..** : aller dans le répertoire parent (les noms « **.** » et « **..** » désignent, respectivement, le répertoire courant et le répertoire parent)
- **file FICHIER** : affiche le type du fichier spécifié dans l'argument.
- **cat FICHIER** : affiche le contenu du fichier spécifié dans l'argument.

Gestion des permissions

Dans les systèmes de la famille Windows, les permissions des fichiers sont spécifiées à l'aide d'ACL (*access control list*). Une ACL est associée à chaque fichier, et chaque entrée de la liste contient :

- Le nom d'une entité de sécurité (*security principal*), p. ex. un·e utilisateur·rice, un ordinateur, ou un groupe.
- Les permissions (create, read, write, modify, delete, etc.) de cette entité sur le fichier.

Dans les systèmes POSIX, bien qu'il soit aussi possible d'utiliser des ACL, par défaut, on spécifie, pour chaque fichier, les permissions ***r****ead*, ***w****rite*, e***x****ecute* pour :

- Le propriétaire (owner)
- Le groupe propriétaire
- Tous les autres (other)

Les commandes **chown** et **chmod** permettent, respectivement, de modifier l'utilisateur·rice ainsi que le groupe propriétaire, et de modifier les permissions d'un fichier.

Lancer l'exécution d'un programme

Par mesure de sécurité (nous y reviendrons), il est nécessaire de spécifier le **chemin** du **fichier exécutable** pour en lancer l'exécution d'un programme dans le shell. Heureusement, il y a des raccourcis.

- Le **chemin** n'est pas nécessairement un **chemin absolu**, cela peut aussi être un **chemin relatif**. Par exemple, si le fichier exécutable se trouve dans le répertoire de travail, alors le chemin devient **./<fichier exécutable>**.
- Si le nom du programme se trouve dans un répertoire listé dans la variable d'environnement **PATH**, alors le nom du programme suffit (la plupart des commandes du shell sont des fichiers exécutables)

La commande « **echo $PATH** » affiche le contenu de la variable d'environnement **PATH**.

Pour pouvoir exécuter un fichier, l'utilisateur·rice doit avoir la permission e***x***ecute sur ce fichier.

Fichier exécutable

Un fichier exécutable est un fichier binaire ou un fichier de texte dont le contenu peut être :

- Chargé et exécuté directement par le système d'exploitation.
- Passé à un **interpréteur** pour être chargé et exécuté.

Un interpréteur est un programme capable d'exécuter un programme. Par exemple, les programmes **java**, **python**, **bash** ou encore **powershell**.

Windows utilise l'extension pour déterminer comment exécuter un fichier. Par exemple, directement pour un fichier **.exe** ou en utilisant PowerShell pour un fichier **.ps1**.

Linux et macOS vérifient les permissions du fichier et lisent sa signature (les premiers octets). Par exemple, si le fichier commence par « 7F 45 4C 46 », il est directement chargé et exécuté, s'il commence par « 23 21 20 » (**#!** ou *shebang*), le chemin de l'interpréteur suit (souvent **/bin/bash**).

De plus, le fichier doit avoir la permission

Programme et processus

Un programme est un texte appelé **code source** écrit dans un langage de programmation ou une séquence de nombres appelée **code objet**, dans le cas d'un langage machine. Ce texte ou cette séquence de nombres décrit un traitement que l'on peut appliquer à des données, ainsi que la structure de ces données.

Un programme réside généralement dans un fichier, qui peut être un **fichier exécutable**.

Pour charger et lancer l'exécution d'un programme dans une machine, on utilise généralement un système d'exploitation.

Comme un système d'exploitation est capable d'exécuter un grand nombre de programmes en même temps, il gère les **programmes en cours d'exécution** en associant à chacun d'eux une structure de données appelée **processus (*process*)**.

Structure d'un processus (I/II)

Un processus contient des informations spécifiques au programme :

- L'adresse à laquelle a été chargé le programme
- L'adresse à laquelle se trouve la pile (*stack*) (pour les variables locales)
- La liste des fils d'exécution (*thread*) associée à ce processus
- Un statut : prêt, en cours d'exécution, bloqué, etc.

Et des informations sur les ressources utilisées par ce programme :

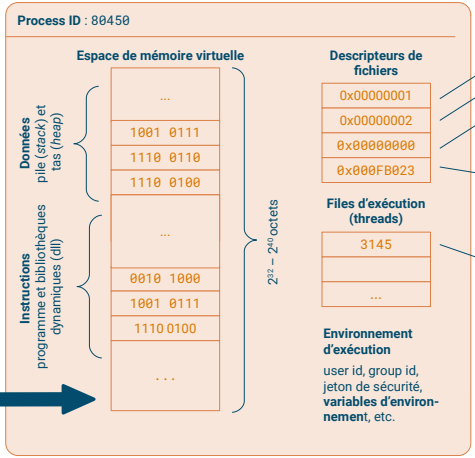
- La liste des descripteurs des fichiers ouverts (y compris *stdin*, *stdout* et *stderr*)
- Les variables d'environnement
- L'identifiant de l'utilisateur qui a lancé le processus.

Structure d'un processus (II/II)

1. L'OS crée une structure de données pour le nouveau processus et lui attribut un identifiant (Process ID).

2. L'OS charge le programme que contient le fichier exécutable (PE, ELF, etc.), dans l'espace de mémoire virtuelle du processus.

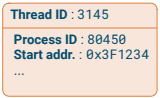
Fichier exécutable du programme



Structure de données du processus

Terminal (stdin, stout, stderr)

Fichier



3. L'OS crée un fil d'exécution (thread) pour processus avec l'adresse du point d'entrée (procédure main) comme adresse de départ et passe le thread à l'ordonnanceur.

Processus parent et processus enfant

À l'exception du tout premier processus qui est créé directement par le noyau, un processus est toujours créé par un autre processus.

Le processus qui en crée un nouveau est appelé **processus parent (*parent process*)**. Le processus créé est appelé **processus enfant (*child process*)**.

Lors de sa création, le processus est essentiellement une copie du processus parent. Il a donc accès aux mêmes ressources que le parent, notamment, les mêmes **descripteurs de fichiers**, dont *stdin*, *stdout* et *stderr*, et les même **variables d'environnement**.

En revanche, pour éviter des conflits, les processus sont isolés les uns des autres. Il ne partage donc pas l'espace de mémoire, et les informations spécifiques au programme correspondent à celui qui a été lancé, qu'il s'agisse ou non du même que celui du processus parent.

Plan

Conclusion

Nous voilà équipés pour aborder le sujet du module : les scripts.

Les raisons pour lesquelles nous avons passé autant de temps sur différents aspects des systèmes d'exploitation et, notamment, sur les interfaces en ligne de commande CLI et le terminal :

- Le module porte sur la réalisation de script dans l'**administration système**
- Contrairement aux interfaces graphiques (GUI), les interfaces en ligne de commande (CLI) sont « naturellement » scriptables, nous verrons comment.
- Pour utiliser un programme avec une interface en ligne de commande, on doit utiliser un terminal. Il est donc important de bien comprendre ce dont il s'agit.
- Un fichier qui contient un script est un fichier exécutable, et l'exécution du script crée un processus enfant. Il est donc important de comprendre les liens qui existent entre processus enfant et processus parent.