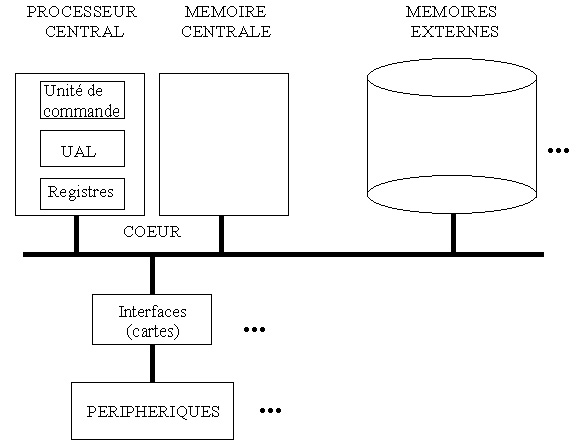
**453---SYSTEME D'EXPLOITATION - e-learning - Séquence 4**

**Préliminaire : définition d’un système d’exploitation**

Avant d’utiliser Debian GNU/Linux dans la séquence 5, un peu de lecture concernant la notion de système d’exploitation, en particulier le système Linux.

**0. Préliminaire : définition d’un système d’exploitation**



*Vision matérielle simplifiée d’un ordinateur*

Un ordinateur monoprocesseur est composé, dans son « cœur », d’un processeur central, lui-même composé d’une unité de commande, d’une unité arithmético-logique (UAL), où sont exécutées les opérations binaires, et de registres, d’une mémoire centrale, et bien souvent d’autres mémoires intermédiaires.

Toutes les informations manipulées par les utilisateurs (images, sons, textes, etc.), afin de pouvoir être traitées par la machine, sont décomposées en séquences d’informations élémentaires, représentées par des suites de chiffres binaires (0 ou 1), appelés « bits » (*binary digits*), structurées généralement en séquences d’« octets » ou de « *bytes* » (suites de 8 bits).

Les informations manipulées par les utilisateurs (les « données »), ainsi que les « programmes », sont rangés dans des fichiers, eux-mêmes regroupés en dossiers, conservés sur des mémoires externes (disques magnétiques, bandes magnétiques, disques optiques ou magnéto-optiques, etc.).

En outre, les utilisateurs communiquent avec la machine via des périphériques, reliés au système au moyen d’interfaces (cartes de périphériques).

Un périphérique d’« **entrée** » permet à un utilisateur de donner de l’information au coeur du système. Un périphérique de « **sortie** » permet à un utilisateur de recevoir de l’information du coeur du système.

Bien sûr, certains périphériques sont à la fois d’entrée et de sortie. C’est le cas par exemple des « écrans tactiles ». Les mémoires externes sur lesquelles on peut lire et écrire peuvent aussi être considérées comme faisant partie de cette catégorie. Les *lecteurs* de DVD ou de CD-ROM (*Compact Disk - Read Only Memory*) n’en font évidemment pas partie. Par contre, les « graveurs » en font partie.

Comme périphériques d’entrée/sortie, il faut encore mentionner les interfaces permettant à un ordinateur d’être relié à un réseau. Il peut s’agir d’une interface purement digitale (traitant des signaux binaires) ou d’un modem (modulateur-démodulateur : interface digitale-analogique réalisant la conversion des signaux binaires en signaux sinusoïdaux et vice versa).

**0.1. Structure en couches logicielles d’un ordinateur**

Un ordinateur est une machine capable de résoudre des problèmes en appliquant des instructions préalablement définies. La suite des instructions décrivant la façon dont l’ordinateur doit effectuer un certain travail est appelée programme. Il s’agit des instructions qui peuvent être « exécutées » par l’unité de contrôle de l’ordinateur.

Les circuits électroniques de chaque ordinateur ne pouvant reconnaître et exécuter directement qu’un nombre très limité d’instructions, tout programme doit être, avant son exécution, converti pour n’être exprimé qu’avec ces instructions. L’ensemble des instructions exécutables directement par un ordinateur forme un langage qui permet en fin de compte aux gens de communiquer avec cet ordinateur. C’est ce qu’on appelle le « langage machine ». Lorsqu’on conçoit un nouvel ordinateur, il convient de choisir les instructions qui formeront son langage machine. En général on cherche à ce que ces instructions soient les plus simples possible, compte tenu des performances attendues de la machine, afin de réduire la complexité (et donc le coût) des circuits électroniques nécessaires. Le problème est que ces langages machine sont alors si primitifs qu’il est extrêmement pénible et fastidieux de les utiliser.

Il y a deux façons de résoudre ce problème qui, toutes les deux, visent à construire un nouveau jeu d’instructions **p**lus **p**ratique à utiliser que le langage **m**achine. L’ensemble de ces nouvelles instructions forme un langage, que nous appellerons **LPP**, de la même façon que l’ensemble des instructions machine forme le langage appelé **LM**. Les deux techniques qui existent diffèrent suivant la manière dont les programmes écrits en **LPP** sont traités par l’ordinateur qui, en dernière analyse, ne peut exécuter que des programmes écrits dans son langage machine **LM**.

La première façon d’exécuter un programme écrit en **LPP** est de remplacer dans un premier temps chaque instruction de ce programme par la suite d’instructions en **LM** qui est équivalente. On obtient ainsi un nouveau programme entièrement écrit en **LM**. L’ordinateur exécute alors ce nouveau programme en **LM** et non pas l’ancien programme en **LPP**. Cette technique est appelée traduction ou compilation, et le programme qui traduit chaque instruction **LPP** en séquence **LM** s’appelle un compilateur. Le résultat de la traduction réside dans des fichiers exécutables.

La seconde façon est d’écrire un programme capable, après avoir examiné chaque instruction d’un programme en **LPP**, d’exécuter directement la séquence d’instructions en **LM** équivalente. Cette technique, avec laquelle on n’a pas besoin de générer tout un programme en **LM**, est appelée interprétation, et le programme capable de réaliser ce traitement s’appelle un interpréteur.

Traduction et interprétation se ressemblent beaucoup. Dans chaque cas toute instruction en **LPP** est finalement convertie en une suite équivalente d’instructions en **LM**. Mais il faut noter que dans le cas de la traduction tout le programme en **LPP** est d’abord converti en un programme en **LM** ; puis le programme en **LPP** « disparaît » et c’est le programme en **LM** qui est exécuté. En revanche, avec l’interprétation chaque instruction du programme en **LPP** est analysée puis immédiatement exécutée. On n’obtient donc pas de programme traduit. Ces deux méthodes sont toutes deux très utilisées.

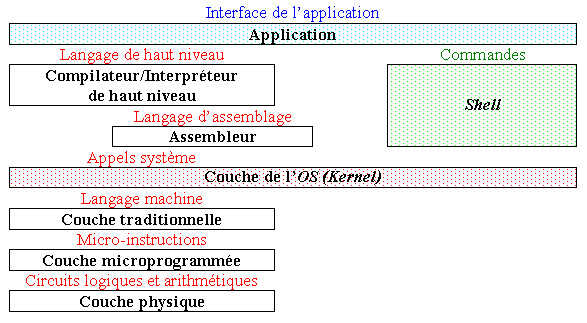
Plutôt que de raisonner en termes de traduction ou d’interprétation, il est souvent plus facile d’imaginer l’existence d’un ordinateur hypothétique, ce qu’on appelle une machine virtuelle, dont le langage machine est **LPP**. Si on pouvait construire réellement cette machine à un coût raisonnable, on n’aurait pas besoin du langage **LM** ni d’une machine capable d’exécuter ce langage **LM**. Les gens écriraient tout simplement leurs programmes en **LPP** et la machine exécuterait directement ces programmes. Bien qu’il soit trop coûteux de construire une telle machine, on peut néanmoins écrire des programmes en **LPP** à condition que ces programmes puissent être traduits ou interprétés par un ordinateur (réel celui-ci) dont le langage est **LM**. En d’autres termes, on peut écrire des programmes pour des machines virtuelles comme si elles existaient réellement.

Pour que la traduction ou l’interprétation reste assez simple il faut que les langages **LM** et **LPP** ne soient pas trop différents. Ceci implique que très souvent **LPP**, bien que « meilleur » que **LM**, n’est pas le langage idéal pour l’écriture d’un programme. Ceci est quelque peu contradictoire avec l’idée même qui avait présidé à l’élaboration de **LPP** : faire un langage plus simple et plus pratique à utiliser par des êtres humains que **LM**, qui était plutôt fait pour la machine.

On peut être alors amené à définir un **n**ouvel ensemble d’instructions plus proches de l’utilisateur final et donc moins dépendantes de la machine que celles de **LPP**. Ces nouvelles instructions forment un langage **LN** à l’aide duquel, comme on l’a vu plus haut, on pourra écrire des programmes comme si une machine basée sur **LN** existait. Ces programmes seront, bien sûr, traduits en **LPP** par un compilateur ou interprétés par un interpréteur.

On peut ainsi concevoir toute une série ou « hiérarchie » de langages, chacun étant un peu plus pratique que son prédécesseur, jusqu’à ce qu’on en obtienne un jugé « convenable ». Chaque langage s’appuie sur son prédécesseur de telle sorte qu’on peut voir un ordinateur comme un empilement de couches ou de niveaux. Le langage « du bas » est le plus simple, celui « du haut » est le plus complexe.

Un ordinateur composé de n couches peut être vu comme n machines virtuelles distinctes, chaque machine virtuelle ayant son propre langage.



*Vision logique simplifiée d’un ordinateur*

La plupart des programmeurs travaillant sur une machine virtuelle de niveau n ne sont intéressés que par le niveau supérieur, celui qui ressemble le moins au langage machine du bas niveau. Cependant, les gens qui s’intéressent au fonctionnement d’un ordinateur doivent étudier tous les niveaux. Ceux qui veulent concevoir de nouvelles machines ou de nouvelles couches (c’est-à-dire de nouvelles machines virtuelles) doivent aussi connaître d’autres couches que les couches supérieures.

Le niveau 0, le premier à intéresser l’informaticien, est appelé niveau physique. Les objets manipulés à ce niveau sont des portes qui, bien que fabriquées à l’aide de composants analogiques comme les transistors, peuvent être vues comme des composants logiques. Chaque porte dispose d’une ou plusieurs entrées logiques (signaux représentant 0 et 1) et rend comme résultat une fonction simple de ces entrées comme ET ou OU. Pour être complet, nous devrions mentionner l’existence d’un niveau encore inférieur au niveau 0. Ce niveau, appelé « niveau composant », est du domaine du génie électrique. Ce qui est manipulé à ce niveau, ce sont des transistors qui sont en quelque sorte les « atomes » des concepteurs de machines. Bien sûr, on pourrait aussi parler du fonctionnement des transistors, mais cela nous entraînerait dans la physique du solide !

A la différence du niveau 0 dans lequel il n’y a pas trace de programme (ou suite d’instructions à exécuter), le niveau 1 comprend un programme, appelé microprogramme, qui se situe généralement dans des mémoires mortes (ROM - Read Only Memory) et dont le travail est d’interpréter les instructions du niveau supérieur. Nous appellerons ce niveau le niveau microprogrammé. Deux ordinateurs différents n’ont jamais exactement des niveaux microprogrammés identiques. Peu de machines disposent à ce niveau de plus de vingt instructions et la plupart de ces instructions consistent à transférer des données ou à exécuter des tests très simples.

Chaque microprogramme définit implicitement un langage de niveau 2 (et une machine virtuelle dont le langage machine est ce langage). Toutes les machines de niveau 2 ont beaucoup de points communs, même si ces machines sont de constructeurs différents. Ce niveau sera appelé niveau machine traditionnel. Les constructeurs d’ordinateurs proposent avec chaque machine qu’ils vendent un manuel souvent intitulé « Manuel de référence du langage machine » ou quelque chose d’approchant. Ces manuels se rapportent au niveau 2 (machine virtuelle) et non au niveau 1. Le jeu d’instructions machine qui y est décrit (entre 50 et 300 instructions) est l’ensemble des instructions interprétées par le microprogramme et non pas l’ensemble des instructions exécutables directement par le matériel.

Notons cependant que certains ordinateurs, et surtout les plus anciens, ne disposent pas de couche microprogrammée. Sur ces machines, les instructions de niveau 2 sont traitées directement par le matériel (niveau 0), sans intervention d’un niveau 1. Nous ne tiendrons pas compte de ces exceptions et nous dirons que le niveau machine conventionnel est le niveau 2.

Le troisième niveau, appelé système d’exploitation (*Operating System*), est souvent un niveau hybride. La plupart des instructions de son langage figurent également dans le langage du niveau 2. Il n’y a en effet aucune raison pour qu’une instruction d’un niveau ne puisse être présente à un autre niveau. On y trouve également des instructions spécifiques à ce niveau (dites « de niveau système d’exploitation »), une organisation de la mémoire différente, la capacité d’exécuter plusieurs programmes en parallèle, etc. Il existe plus de différences entre les machines de niveau 3 qu’entre les machines de niveau 1 ou 2. Les instructions du niveau 3 qui sont identiques à des instructions du niveau 2 sont traitées directement par le niveau inférieur et non par le système d’exploitation. C’est en cela que ce niveau est hybride.

Il y a un fossé entre les niveaux 3 et 4. Les quatre niveaux inférieurs ne concernent pas directement le programmeur moyen. Ils sont là principalement pour supporter les traducteurs et interpréteurs dont les niveaux supérieurs ont besoin et sont écrits par des spécialistes de la réalisation de machines virtuelles : les programmeurs système. Les niveaux 4 et au-dessus concernent le programmeur d’applications qui a un problème concret à résoudre.

Autre caractéristique qui apparaît au niveau 4 : la méthode de support des niveaux supérieurs. Les niveaux 2 et 3 sont toujours interprétés. Les niveaux 4 et 5 sont souvent, mais pas toujours, associés à des compilateurs (traducteurs).

Il y a d’ailleurs une autre différence entre les niveaux 1, 2 et 3 d’un côté et les niveaux supérieurs de l’autre : c’est la nature du langage fourni. Les langages « machine » des niveaux 1, 2 et, originellement, 3 sont numériques, donc très difficiles à utiliser puisqu’ils consistent en de longues séries de nombres. Au contraire, à partir du niveau 4, on trouve des langages qui contiennent des mots ou des abréviations beaucoup plus compréhensibles par les êtres humains (mais pas par la machine).

Le niveau 4 est appelé assembleur (il s’agit d’un traducteur). Le langage d’assemblage est une forme symbolique d’un des langages sous-jacents. Ce niveau permet d’écrire des programmes pour les niveaux 1, 2 et 3 dans une forme moins déplaisante. Les programmes en langage d’assemblage sont d’abord traduits en langage de niveau 1, 2 ou 3 puis sont interprétés par la machine virtuelle ou réelle correspondante. Si ce langage d’assemblage était autrefois très important, on peut dire qu’aujourd’hui son importance est bien moindre. En effet, au cours du temps, de nouvelles couches ont été insérées dans la structure de manière à assurer une certaine « portabilité » des développements de haut niveau. C’est par exemple l’objectif des plates-formes « JAVA » ou « .NET ».

Au niveau 5 on trouve des compilateurs ou des interpréteurs de langages conçus pour être utilisés par des programmeurs d’applications. Ces langages, souvent appelés langages de haut niveau, sont extrêmement nombreux : il en existe des centaines.

Les niveaux 6 et au-dessus sont des ensembles de programmes conçus pour créer des machines virtuelles destinées à traiter des applications déterminées.

En résumé, on peut dire qu’un ordinateur peut être vu comme une suite de couches, chaque couche englobant toutes les couches précédentes. Une couche représente un certain niveau d’abstraction et comporte divers objets et opérations sur ces objets. L’ensemble des types de données, des opérations et des caractéristiques de chaque niveau s’appelle l’architecture de ce niveau.

**0.2. Logiciel système**

On peut dire que les logiciels (les programmes) se répartissent en deux grandes catégories : les programmes système qui permettent le fonctionnement de l’ordinateur, et les programmes d’application qui résolvent les problèmes des utilisateurs.

Le système d’exploitation (*Operating System*) est le plus important des programmes système. Il contrôle les ressources de l’ordinateur et fournit la base sur laquelle seront construits les programmes d’application.

Le reste du logiciel système se trouve au-dessus du système d’exploitation. On y trouve l’interpréteur de commandes (*Shell*), les éditeurs, les compilateurs et d’autres programmes qui ne dépendent pas non plus des programmes d’application. Ces programmes ne font pas partie du système d’exploitation, même s’ils sont fournis par les constructeurs d’ordinateurs.

Le système d’exploitation est la partie du logiciel système qui fonctionne en mode noyau ou mode superviseur. Elle est protégée par le matériel contre les erreurs de manipulation des utilisateurs. Les éditeurs et les compilateurs fonctionnent en mode utilisateur. Si un utilisateur n’apprécie pas un compilateur donné, il peut, s’il le souhaite, écrire son propre compilateur ; en revanche, il ne peut pas écrire son propre gestionnaire des interruptions du disque car celui-ci fait partie du système d’exploitation et est protégé par le matériel contre les tentatives des utilisateurs pour le modifier.

Enfin, au-dessus des programmes système, on trouve les programmes d’application. Ces programmes sont écrits par les utilisateurs ou par les éditeurs de logiciels pour résoudre des problèmes spécifiques tels que le traitement des données commerciales, les calculs scientifiques ou les jeux.

**0.3. A quoi sert un système d’exploitation ?**

L’interface entre le système d’exploitation et les programmes de l’utilisateur est constituée d’un ensemble d’« instructions étendues » fournies par le système d’exploitation. Ces instructions étendues sont qualifiées d’appels système. Les appels système créent, détruisent et utilisent divers objets logiciels gérés par le système d’exploitation. Les processus et les fichiers sont les plus importants de ces objets.

Fondamentalement, un système d’exploitation rend des services en tant que machine étendue et en tant que gestionnaire de ressources. Cette double vocation d’un système d’exploitation est développée ci-après.

**0.3.1. Le système d’exploitation en tant que machine étendue**

Un ordinateur se compose d’un ou de plusieurs processeurs (unités de contrôle), d’une mémoire principale, d’horloges, de mémoires secondaires ou externes, de terminaux, et d’interfaces de connexion à d’autres périphériques d’entrées/sorties (E/S) et à des réseaux. Il s’agit d’un système fort complexe. L’écriture de programmes qui prennent en compte tous ces éléments et les gèrent correctement est un travail extrêmement difficile.

Au niveau du langage machine, les périphériques d’E/S sont contrôlés en chargeant des valeurs spécifiques dans des registres spéciaux, les registres des périphériques. On peut, par exemple, mettre un disque en mode lecture/écriture en chargeant ses registres avec l’adresse du disque, l’adresse de la mémoire principale, le nombre d’octets à transférer et le sens de transfert (lecture ou écriture). Dans la pratique, il faut de nombreux autres paramètres et le code retourné par l’unité de disque à la fin de l’opération est très complexe. De plus, pour de nombreux périphériques d’E/S, le temps est un facteur important lors de la programmation.

A titre d’exemple, la programmation des E/S des lecteurs de disquettes au moyen du contrôleur NEC PD765 (utilisé sur l’IBM PC et de nombreux autres ordinateurs personnels) s’effectue selon 16 commandes qui consistent toutes à charger entre 1 et 9 octets dans un registre de données. Ces commandes permettent de lire et d’écrire des données, de déplacer le bras du lecteur de disquettes, de formater les disquettes ainsi que d’initialiser, tester, restaurer et recalibrer le contrôleur et les lecteurs. Les commandes fondamentales sont READ (lire) et WRITE (écrire), chacune demandant 13 paramètres regroupés dans 9 octets. Ces paramètres spécifient des éléments tels que l’adresse du secteur à lire, le nombre de secteurs par piste, le mode d’enregistrement utilisé sur le support physique, la distance entre deux secteurs consécutifs et ce qu’il faut faire à la rencontre d’une marque d’effacement de données. A la fin de l’opération, le contrôleur retourne 23 champs d’état et d’erreur regroupés dans 7 octets. En outre, le programmeur du lecteur de disquettes doit toujours vérifier si le moteur est allumé ou éteint. S’il est éteint, il faut le mettre en route (et attendre que sa vitesse se stabilise) avant de pouvoir lire ou écrire les données. Il ne faut cependant pas laisser le moteur allumé trop longtemps car cela userait la disquette. Le programmeur système doit donc trouver un compromis entre de longs délais de démarrage et l’usure des disquettes (et les risques de perte de données qui s’ensuivraient).

Bien peu de programmes seraient développés si chaque programmeur devait connaître le fonctionnement détaillé des disques et toutes les erreurs qui peuvent apparaître lors de la lecture d’un bloc de données. Il est donc apparu clairement qu’il fallait trouver un moyen de libérer les programmeurs de la complexité du matériel. Ce moyen, qui a évolué petit à petit, consiste à enrober le matériel d’une couche de logiciel qui gère l’ensemble du système. Il faut en fait présenter au programmeur une interface ou machine virtuelle plus facile à comprendre et à programmer. Cette couche de logiciel est le système d’exploitation.

Le programmeur d’applications souhaite, pour programmer efficacement, une abstraction simple et de haut niveau. Dans le cas des disques, une abstraction typique est que le disque contient des fichiers nommés. Chaque fichier peut alors être ouvert en lecture ou en écriture. Puis il est lu ou écrit et finalement fermé.

Le système d’exploitation est donc le programme qui soustrait le matériel aux regards du programmeur et offre une vue simple et agréable de fichiers nommés qui peuvent être lus ou écrits. Le système d’exploitation masque de la même manière beaucoup d’aspects fastidieux en ce qui concerne les interruptions, les horloges, la gestion de la mémoire et les autres tâches de bas niveau. Dans tous les cas, l’abstraction présentée à l’utilisateur est plus simple et plus facile à utiliser que le matériel sous-jacent.

De ce point de vue, la fonction du système d’exploitation est de présenter à 1’utilisateur l’équivalent d’une machine étendue ou machine virtuelle plus facile à programmer que le matériel.

**0.3.2. Le système d’exploitation en tant que gestionnaire   
 de ressources**

Nous venons de voir le système d’exploitation en tant qu’interface de programmation plus commode. Mais on peut également le considérer comme un gestionnaire des fonctions complexes d’un système lui aussi complexe. Les ordinateurs se composent d’une multitude d’éléments. Dans cette optique, le travail du système d’exploitation consiste à ordonner et à contrôler l’allocation des processeurs aux processus et à gérer leur exécution, à gérer l’allocation des mémoires, à gérer l’accès aux fichiers et aux périphériques d’Entrées/Sorties, pour les différents programmes qui en ont besoin.

Par exemple, si trois programmes qui s’exécutent sur un ordinateur essayent simultanément d’imprimer leurs résultats sur la même imprimante, le système d’exploitation peut transférer les résultats à imprimer dans un fichier tampon sur disque. Lorsqu’un programme se termine, le système d’exploitation peut alors copier ses résultats du fichier où ils ont été sauvegardés vers l’imprimante. Simultanément, un autre programme peut continuer à générer des résultats sans se rendre compte qu’il ne les envoie pas directement à l’imprimante.

Lorsqu’un ordinateur est partagé entre plusieurs utilisateurs, la nécessité de gérer et de protéger toutes les ressources est encore plus évidente. De plus, il est souvent nécessaire de partager des informations entre des personnes qui travaillent ensemble. Le rôle principal du système d’exploitation, de ce point de vue, est de connaître à tout moment l’utilisateur d’une ressource, de gérer les accès à cette ressource, d’en accorder l’usage et d’éviter les conflits d’accès entre différents programmes ou entre différents utilisateurs.

**0.4. Historique des systèmes d’exploitation**

Les systèmes d’exploitation ont évolué au fil des ans, au gré de l’évolution de l’architecture des ordinateurs sur lesquels ils fonctionnent. Il est donc nécessaire de passer en revue les générations successives d’ordinateurs pour en examiner les apports successifs au niveau du fonctionnement des systèmes d’exploitation.

**0.4.1. La première génération (1945-1955) : les tubes à vide et les   
 cartes enfichables**

Vers le milieu des années 40, Howard Aiken de Harvard, John von Neumann de Princeton, Konrad Zuse en Allemagne et d’autres encore ont réussi à construire des machines à calculer au moyen de tubes électroniques. Ces machines étaient énormes, remplissaient des salles entières avec des centaines de tubes et étaient extrêmement lentes et peu fiables.

A cette époque, un seul groupe de personnes concevait, construisait, programmait, utilisait et effectuait la maintenance d’une machine. La programmation se faisait intégralement en langage machine, souvent en reliant électriquement des cartes pour contrôler les fonctions de base de l’ordinateur. Les systèmes d’exploitation étaient alors inconnus. Le mode de fonctionnement consistait à réserver une tranche de temps pour chaque programmeur. Ce dernier se rendait à la salle de l’ordinateur et y insérait son circuit. Pratiquement tous les programmes ne servaient qu’à effectuer des calculs numériques.

Au début des années 50, la procédure s’est améliorée grâce à l’introduction des cartes perforées. Il devint alors possible d’écrire des programmes sur ces cartes et de les faire lire à l’ordinateur plutôt que d’utiliser des cartes électriques à enficher ; le reste de la procédure ne changea pas.

**0.4.2. La deuxième génération (1955-1965) : les transistors et le   
 traitement par lots**

L’introduction du transistor vers le milieu des années 50 a complètement modifié la situation. Les ordinateurs devinrent suffisamment fiables pour être construits et vendus à des clients. Pour la première fois, une séparation nette existait entre les concepteurs, les constructeurs, les opérateurs, les programmeurs et le personnel de maintenance.

Ces machines étaient enfermées dans des pièces dont l’air était conditionné et elles étaient surveillées par des opérateurs professionnels. Seules les grandes compagnies, les administrations ou les universités pouvaient se permettre une telle dépense. Pour lancer un travail (*job*), c’est-à-dire un programme ou un ensemble de programmes, le programmeur devait commencer par écrire le programme en assembleur ou en FORTRAN, puis il devait le mettre sur des cartes perforées. Il apportait ensuite ses cartes à la salle de soumission des travaux et les passait à l’un des opérateurs, qui les donnait à lire à l’ordinateur. Si le compilateur FORTRAN se révélait nécessaire, il fallait également le charger dans l’ordinateur. La majeure partie du temps était perdue en manipulations.

Etant donné le coût élevé des équipements, on chercha à réduire les pertes de temps en adoptant la solution du traitement par lots (*batch processing*). L’idée directrice était de collecter un ensemble de travaux puis de les transférer sur des bandes magnétiques en utilisant pour cela un petit ordinateur (relativement) peu onéreux, comme l’IBM 1401. D’autres machines, bien plus chères, comme l’IBM 7094, étaient utilisées pour les calculs. L’opérateur devait donc y charger un programme spécial (l’ancêtre des systèmes d’exploitation d’aujourd’hui) qui lisait chaque programme successif et l’exécutait, les résultats étant écrits sur une autre bande magnétique au lieu d’être imprimés directement. A la fin de l’ensemble des travaux, l’opérateur changeait les bandes d’entrée et de sortie, puis les résultats pouvaient être imprimés par un IBM 1401 qui fonctionnait de manière autonome.

A cette époque, la structure d’un travail apporté à la salle de soumission des travaux est la suivante : elle commence par une carte $JOB qui spécifie le temps maximal d’exécution exprimé en minutes, le numéro de compte et le nom du programmeur. Puis vient une carte $FORTRAN qui indique au système d’exploitation de charger le compilateur FORTRAN à partir de la bande système. Elle est suivie de cartes contenant le programme à compiler (fichier source), puis d’une carte $LOAD qui indique au système d’exploitation de charger le code objet qui vient d’être généré (les programmes compilés étaient fréquemment écrits dans des fichiers temporaires et devaient être chargés explicitement). Une carte $RUN indique ensuite à l’ordinateur d’exécuter le programme avec les données qui figurent sur les cartes suivantes. Enfin, une carte $END marque la fin du travail. Ces cartes de contrôle, quoique primitives, étaient les précurseurs des langages de contrôle et des interpréteurs de commandes modernes.

Les ordinateurs de la seconde génération étaient surtout utilisés pour les calculs scientifiques et d’ingénierie. Les systèmes d’exploitation les plus connus étaient FMS (*Fortran Monitor System*) et IBSYS, le système d’exploitation d’IBM pour le 7094.

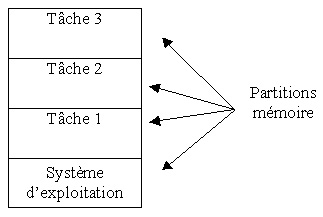
**0.4.3. La troisième génération (1965-1980) : les circuits intégrés,   
 la multiprogrammation, etc.**

Au début des années 60, la plupart des constructeurs d’ordinateurs avaient deux lignes de produits. Il y avait d’une part les gros ordinateurs scientifiques et d’autre part les ordinateurs commerciaux, largement utilisés par les banques et les compagnies d’assurance pour le tri des bandes et l’impression. Le maintien de ces deux lignes de produits différentes était trop coûteux pour les fabricants. De plus, de nombreux utilisateurs demandaient au départ un petit ordinateur, mais souhaitaient par la suite un ordinateur plus puissant.

IBM tenta de résoudre d’un coup ces deux problèmes en introduisant le système OS/360, conçu à la fois pour les calculs scientifiques et commerciaux, donnant naissance à une série d’ordinateurs compatibles entre eux au niveau du logiciel et ne différant qu’en termes de prix et de performance (mémoire maximale, vitesse du processeur, nombre de périphériques d’E/S connectables, etc.). Comme toutes ces machines possédaient, en théorie du moins, la même architecture et le même jeu d’instructions, les programmes développés pour une machine pouvaient s’exécuter sur toutes les autres. La série 360 fut la première ligne d’ordinateurs à utiliser des circuits intégrés qui lui ont permis d’offrir un rapport performance/coût bien supérieur à celui des ordinateurs de la deuxième génération.

Le point fort du concept d’une « famille unique » fut également son point faible. L’idée de départ était que tout logiciel, y compris le système d’exploitation, devait fonctionner sur tous les modèles. Cependant, il devait être efficace sur ces ordinateurs aux vocations souvent si différentes et il devait répondre à des besoins souvent contradictoires. Il en résulta un OS/360 énorme et très complexe. Il était constitué de millions de lignes en assembleur écrites par des centaines de programmeurs. Elles contenaient des centaines de bogues qu’il fallait sans cesse corriger. Chaque nouvelle version corrigeait quelques bogues mais en introduisait aussi des nouveaux, ce qui fait que le nombre de bogues est pratiquement resté constant au cours du temps.

Malgré son énorme taille et ses problèmes, l’OS/360 et les autres systèmes d’exploitation de la troisième génération ont raisonnablement satisfait leurs utilisateurs, en mettant en application quelques techniques qui faisaient défaut aux ordinateurs de la deuxième génération. La multiprogrammation est probablement la plus importante d’entre elles. Pour que le processeur ne reste pas inoccupé pendant les opérations d’E/S, la mémoire a été partitionnée. Chaque partition différente contient un travail ou une « tâche ». Lorsqu’un travail attend la fin d’une E/S, un autre travail peut utiliser le processeur. Si la mémoire principale peut contenir suffisamment de travaux, le processeur peut être utilisé pratiquement en permanence. La présence simultanée de plusieurs travaux en mémoire nécessite des procédures de contrôle pour protéger chaque travail contre les intrusions ou les erreurs des autres. Les ordinateurs de la troisième génération possédaient ces procédures.



La possibilité de transférer les travaux des cartes vers le disque dès leur arrivée dans la salle des ordinateurs constitue un autre apport des systèmes d’exploitation de la troisième génération. Ainsi, dès qu’un travail se termine, le système d’exploitation peut charger, à partir du disque, un nouveau travail dans la partition qui vient de se libérer, puis il peut lancer son exécution. Cette technique s’appelle SPOOL (*Simultaneous Peripheral Operation On Line*). Elle est également utilisée pour les sorties, vers les imprimantes par exemple. Les manipulations de bandes et les ordinateurs auxiliaires moins chers sont devenus ainsi inutiles.

Ensuite, pour que les programmeurs ne doivent plus attendre quelques heures entre le moment de la soumission d’un travail et le moment du retrait des résultats, un système à temps partagé (CTSS ou *Compatible Time Sharing System*) a été mis au point au M.I.T. (Massachusetts Institute of Technology). Il s’agit d’une variante de la multiprogrammation dans laquelle chaque utilisateur possède un terminal en ligne, le processeur étant réparti entre les tâches actives selon un algorithme d’attribution de tour de rôle, le système étant fondé sur l’énorme différence entre les temps de réaction typiques d’un cerveau humain et d’un processeur central. L’ordinateur peut donc répondre rapidement à un nombre important d’utilisateurs et peut même continuer un traitement par lots en arrière-plan lorsque le processeur est libre. Le partage du temps ne s’est vraiment développé qu’avec la généralisation des protections matérielles.

Le M.I.T., les laboratoires Bell et General Electric se sont dès lors attelés au développement d’une machine pouvant supporter la connexion simultanée de plusieurs centaines d’utilisateurs. Le système d’exploitation s’appelait MULTICS (*MULTiplexed Information and Computing Service*). Depuis lors, le concept d’un gros ordinateur central s’est effrité. Néanmoins, MULTICS a beaucoup influencé les systèmes ultérieurs. Autre événement majeur de la troisième génération : le développement phénoménal des mini-ordinateurs, avec la série des PDP de DEC, dont le plus performant était le PDP-11. L’un des informaticiens des laboratoires Bell, Ken Thompson, qui avait participé au projet MULTICS, se mit à écrire sur un PDP-7 une version simplifiée de MULTICS, appelée d’abord UNICS et ensuite UNIX. Depuis, le système d’exploitation UNIX a bien évolué et s’est répandu dans le marché des   
mini-ordinateurs, des stations de travail et des serveurs, sous diverses versions, jusqu’à son « descendant » LINUX.

**0.4.4. La quatrième génération (depuis 1980) : les LSI et VLSI,   
 les ordinateurs personnels, etc.**

L’époque de l’ordinateur personnel est venue avec le développement des circuits LSI (*Large Scale Integration*), puis VLSI (*Very Large Scale Integration*). Au niveau de l’architecture, les ordinateurs personnels ne sont pas sensiblement différents des   
mini-ordinateurs. La différence se situe au niveau du prix et de la performance. Le mini-ordinateur a permis aux différents départements des sociétés et des universités d’avoir leur propre ordinateur. Le micro-ordinateur permet à chacun d’avoir son ordinateur. Quant aux « stations de travail », ce ne sont en fait que des ordinateurs personnels puissants. Ces stations sont habituellement connectées à un réseau local (*LAN = Local Area Network*), concept qui s’est généralisé depuis 1980.

La large disponibilité de puissance de calcul, particulièrement lorsqu’elle est associée à une forte interactivité et à un excellent graphisme, a entraîné le développement d’une importante industrie de production de logiciels pour les ordinateurs individuels.

Des systèmes d’exploitation tels que Microsoft MS-DOS puis Microsoft Windows ont dominé le marché des ordinateurs personnels, tandis que UNIX, puis les versions suivantes de Microsoft Windows ainsi que LINUX ont dominé celui des stations de travail et des serveurs. Les dernières versions de MS-DOS et Windows ont intégré des caractéristiques provenant d’UNIX, ce qui n’est guère surprenant lorsqu’on sait que Microsoft est un des grands fournisseurs des versions successives d’UNIX. Les « stations de travail » (*workstations & servers*) ont progressivement acquis la puissance d’un mini-ordinateur, pour finir par être dotées d’une puissance même supérieure. Il est donc logique qu’elles utilisent un système d’exploitation initialement conçu pour les mini-ordinateurs.

Depuis le milieu des années 80, on assiste au développement d’interfaces utilisateur conviviales, ainsi qu’à l’élaboration des réseaux d’ordinateurs individuels. En réalité, les idées qui sont à la base de ces développements sont nées il y a bien plus longtemps, dans des lieux tels que le « Xerox Parc ».

Le Xerox Palo Alto Research Center est fondé en 1970 dans le parc industriel de l’université de Stanford, en plein coeur d’une bande de terrain coincée entre la baie de San Francisco et la chaîne côtière de Santa Cruz. A l’époque, ce territoire d’une cinquantaine de kilomètres de longueur, largement à l’état de friche, n’est pas encore baptisé la « Silicon Valley ». Et le Parc entame alors sa mission : imaginer une société du futur, que les responsables de Xerox ont l’intelligence de pressentir sans papier mais avec un ordinateur sur chaque bureau.

Alors que le PC n’existe pas encore, pas plus que Microsoft (1975) ou Apple (1976), le fabricant de photocopieuses imagine déjà les effets d’une société basée sur la communication électronique. Et les chercheurs de haut vol rassemblés au Parc reçoivent pour ambitieuse mission de provoquer cette évolution, afin de mieux l’accompagner et surtout d’en prévenir les conséquences pour Xerox. Concrètement, ils doivent tenter de rendre les ordinateurs faciles à utiliser, ce qui les amènera en réalité à réinventer la relation homme-machine, sur base de leurs innovations techniques mais aussi sur base des travaux de certains psychologues comme le suisse Piaget ou l’américain Bruner.

Il faut préciser qu’au moment de se lancer dans cette entreprise, la compagnie Xerox avait elle-même été marquée par les idées d’un certain Doug Engelbart. Dès les années 50, cet ingénieur visionnaire prétendit en effet qu’un jour le commun des mortels s’installerait devant un ordinateur pour y manipuler textes et graphiques et parvint à convaincre le Pentagone de lui fournir les moyens de réaliser un prototype matérialisant ses concepts. C’est en 1968, lors d’une conférence informatique se tenant à San Francisco, que l’ingénieur présenta son appareil : une console branchée, par le biais d’un réseau de télévision, sur l’ordinateur du Stanford Research Institute. A l’aide d’un écran, d’un clavier, d’un dispositif de pointage (un ancêtre de la souris) et d’un système complexe de clefs à bascule lui permettant de fournir des instructions à l’ordinateur, Engelbart fit d’un seul coup la démonstration du premier traitement de texte et du premier système fonctionnel d’hypertexte.

Mais à l’époque, les éminences n’ont pas encore foi en l’informatique personnelle, et restent largement insensibles à cette démonstration. Critiqué pour le coût de ses travaux, Engelbart doit mettre la clef de son laboratoire sous le paillasson.

Plus tard, Xerox embauche une partie des ingénieurs d’Engelbart ainsi que d’autres, et dès 1973, le fabricant de photocopieuses dispose d’un ordinateur qui influencera tous les autres. Cet ordinateur, baptisé « Alto », dispose d’une souris à trois boutons, d’une unité à cassettes et est connecté, via un réseau local Ethernet, à d’autres ordinateurs ainsi qu’à une imprimante laser (autant d’inventions dues en partie à Xerox). Quant à l’interface graphique, elle met en scène des icônes et des fenêtres à menus déroulants, tandis que Smalltalk, le langage de programmation, est orienté objet et repose sur la notion d’informatique distribuée.

Confronté à la concurrence des Japonais sur le marché de la photocopie, Xerox commet l’énorme bourde de ne pas donner suite à ces inventions. L’histoire démontre que Xerox aurait été bien avisé de se lancer dans ce créneau commercial encore vierge à l’époque. Car en effet Steve Wozniak et Steve Jobs ont été admirer l’ordinateur Alto. Les fondateurs d’Apple s’en inspireront en créant le « Lisa » en 1983 puis, l’année suivante, leur premier Macintosh, lequel inspira à son tour un certain Bill Gates ...

Aujourd’hui, les réseaux d’ordinateurs individuels peuvent fonctionner sous des systèmes d’exploitation en réseau ou sous des systèmes d’exploitation distribués.

Sous un système d’exploitation en réseau, les utilisateurs connaissent l’existence des différents ordinateurs, peuvent se connecter sur une machine distante et transférer des fichiers d’une machine à une autre. Chaque ordinateur fonctionne sous son propre système d’exploitation.

A l’opposé, un système d’exploitation distribué apparaît aux yeux de ses utilisateurs comme une seule machine, même si ce n’est pas le cas. Dans un vrai système distribué, les utilisateurs n’ont pas à savoir où se trouvent leurs fichiers et où leurs programmes sont exécutés. Le système d’exploitation doit s’acquitter de sa tâche de manière transparente pour l’utilisateur.

Par contre, un système d’exploitation en réseau ne diffère pas fondamentalement d’un système classique. Il requiert évidemment une interface réseau et son logiciel de contrôle de bas niveau, ainsi que des programmes qui permettent une connexion distante et un accès aux fichiers qui se trouvent sur les différentes machines. Mais ces ajouts ne modifient pas la structure fondamentale du système d’exploitation.

**0.5. Linux**

Voici des sites à consulter :

<https://fr.wikipedia.org/wiki/Linux>

<http://linuxfr.org/>

Linux est le nom donné à tout système d’exploitation libre (à toute « distribution ») fonctionnant avec le noyau Linux. C’est un développement libre du système UNIX respectant les spécifications POSIX.

Ce système est né de la rencontre entre le mouvement du logiciel libre et le modèle de développement collaboratif et décentralisé via Internet. Son nom vient du créateur du noyau Linux, Linus Torvalds.

La Free Software Foundation utilise plutôt le nom GNU/Linux, étant donné que depuis 1992, le noyau est distribué sous la Licence publique générale GNU (GNU GPL, écrite par Richard Stallman).

En ce qui concerne l’identification du noyau, toute version PAIRE (par exemple 2.6) est une version de noyau STABLE, tandis que toute version IMPAIRE (par exemple 2.5) est une version de noyau INSTABLE (en phase de test).

Les distributions Debian GNU/Linux, Ubuntu, etc. utilisent les paquets au format « .deb », tandis que les distributions Red Hat Enterprise Linux, CentOS, etc. utilisent les paquets au format « .rpm ».

Une connexion au système Linux (Ouverture de session ou Logging on), dans l’interface graphique ou dans une interface texte, nécessite la saisie de deux informations : Username et Password.

Pour administrer le système, il faut se connecter en « superutilisateur » (Username = « root »), qui a tous les droits sur le système, et non en utilisateur ordinaire.

La totalité des tâches d’administration du système (station de travail ou serveur) peut être réalisée par des commandes en mode texte.

Les tâches les plus simples peuvent être accomplies via des outils en mode graphique. Les outils sont propres à chaque distribution. Cependant, certains outils, tels que linuxconf ou Webmin, sont multi distribution.