Brève histoire des systèmes d'exploitation

Sacha Krakowiak [1, 2]

> *"An operating system is a collection of things that don't fit into a language. There shouldn't be one."*
>
> « Un système d'exploitation, c'est une collection de choses qui ne tiennent pas dans un langage. Ça ne devrait pas exister. »
>
> Dan Ingalls, *Design Principles Behind Smalltalk*, *Byte Magazine*, August 1981.

Dans un monde idéal, tel que l'imagine l'auteur de la citation ci-dessus, les constructions d'un langage de programmation pourraient être directement réalisées sur un ordinateur, grâce à une couche appropriée de logiciel. Cette vue reflète l'idée d'une machine dédiée à un langage unique, telle qu'elle était mise en œuvre dans Smalltalk. En fait, à partir du moment où il existe plusieurs langages ou environnements de programmation, on peut identifier diverses fonctions d'intendance qu'il serait sans intérêt, et peu économique, de reproduire pour chacun de ces outils. Le logiciel qui réalise cet ensemble de fonctions (gestion de fichiers, entrées-sorties, gestion d'activités parallèles, sécurité, etc.) est le système d'exploitation. Au contact de la machine physique, il transforme celle-ci en une machine idéale (virtuelle) fournissant à la couche supérieure les services d'intendance jugés adéquats ; cette dernière couche utilise ces services pour permettre aux utilisateurs de mettre en

1. Professeur émérite, université Grenoble Alpes. http://lig-membres.imag.fr/krakowia/

2. L'essentiel de la matière de cet article est extrait de deux articles publiés par l'auteur sur le site d'Interstices (https://interstices.info/) : « La naissance des systèmes d'exploitation » (avec Jacques Mossière) et « Les débuts d'une approche scientifique des systèmes d'exploitation », avec l'aimable autorisation de la rédaction d'Interstices.

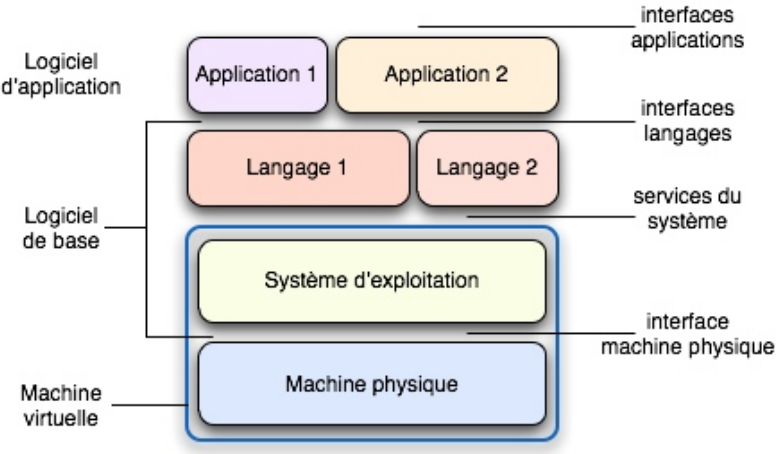

FIGURE 1. Organisation d'un système informatique

œuvre les langages dont ils ont besoin. Ces langages, à leur tour, servent à réaliser les applications. C'est ce que résume la figure 1.

La réalité est plus complexe. D'une part, le nombre de couches est plus grand et la séparation entre elles souvent moins nette ; il peut aussi être plus réduit, avec une ou des applications s'exécutant directement en lieu et place du système. D'autre part, le schéma décrit est celui d'un utilisateur unique. L'ordinateur peut aussi être partagé entre plusieurs utilisateurs, et c'est au système d'exploitation de créer pour chacun d'eux sa propre machine virtuelle, indépendante de celles des autres mais pouvant partager des informations avec elles.

La place des systèmes d'exploitation au contact direct de la machine physique implique une double interaction : d'une part les systèmes évoluent pour suivre les innovations du matériel induites par celles de la technologie ; et d'autre part les besoins des systèmes, eux-mêmes venant des exigences des utilisateurs, imposent des transformations de l'architecture des machines. La suite de cet article fournit quelques exemples de cette double interaction.

Le rôle du système d'exploitation comme transformateur d'interface en implique un autre : celui de gérant des ressources de la machine physique : processeurs, mémoires, organes de communication. Gérer les ressources consiste à les attribuer à tout moment aux activités qui en ont besoin, en prévenant ou en résolvant les conflits éventuels et en respectant les priorités imposées. Il faut également gérer les ressources immatérielles (programmes et données) en permettant si nécessaire leur partage entre utilisateurs et en assurant leur pérennité et leur sécurité.

Cette double fonction (fournir une interface « commode », allouer les ressources) fait que la conception d'un système d'exploitation est soumise à deux exigences : la première fonction privilégie le confort des utilisateurs (on peut parler d'ergonomie) ; la seconde met l'accent sur l'efficacité, l'emploi optimal des ressources (il s'agit là d'économie). L'histoire des débuts des systèmes d'exploitation, développée dans la section 1, est celle du conflit entre ces deux exigences, qui, dans le contexte technique de l'époque, étaient contradictoires.

Le reste de l'article est organisé comme suit. La section 2 décrit la transition du domaine des systèmes d'exploitation depuis un savoir-faire technique vers une discipline scientifique. La section 3 montre la relative stabilisation du domaine après une évolution mouvementée. La section 4 conclut par l'examen de quelques défis actuels

1. Les débuts des systèmes d'exploitation

Les premières décennies de l'histoire des systèmes d'exploitation voient un mouvement de balancier entre les objectifs d'économie et d'ergonomie, privilégiant successivement l'un et l'autre, avant l'arbitrage final, permis par l'évolution technique, en faveur de l'ergonomie. C'est ce qu'illustre la figure 2, qui schématise quelques étapes marquantes des débuts des systèmes [3].

Les détails de cette évolution sont examinés dans les sections qui suivent.

1.1. La préhistoire

Les premiers ordinateurs n'avaient pas à proprement parler de système d'exploitation, et cette situation dura de 1949 (premières machines à programme enregistré) jusqu'en 1956 (premier moniteur de traitement par lots). En effet, les langages initialement utilisés étaient très proches de la machine physique : code binaire, assembleurs primitifs, collections de sous-programmes. D'autre part, le partage de la machine se faisait simplement par réservation : chaque utilisateur avait la machine pour lui seul pendant une tranche de temps déterminée. En ce sens, l'exploitation des ordinateurs privilégiait le confort des utilisateurs, donc le côté « ergonomie ».

On peut néanmoins noter les premiers embryons de mécanismes d'assistance à l'utilisation des ordinateurs. L'EDSAC de Cambridge (première machine à programme enregistré, avec le Manchester Mark-1), utilisait une forme rudimentaire de langage assembleur, où les instructions étaient écrites sous forme symbolique et les adresses (absolues) sous forme décimale. Les programmes, enregistrés sur ruban perforé, étaient traduits en binaire et placés en mémoire par un mécanisme appelé « ordres initiaux », lui-même enregistré sur une mémoire externe en lecture

3. Nous utilisons souvent le terme de « système » à la place de « système d'exploitation », lorsqu'il n'y a pas ambiguïté.

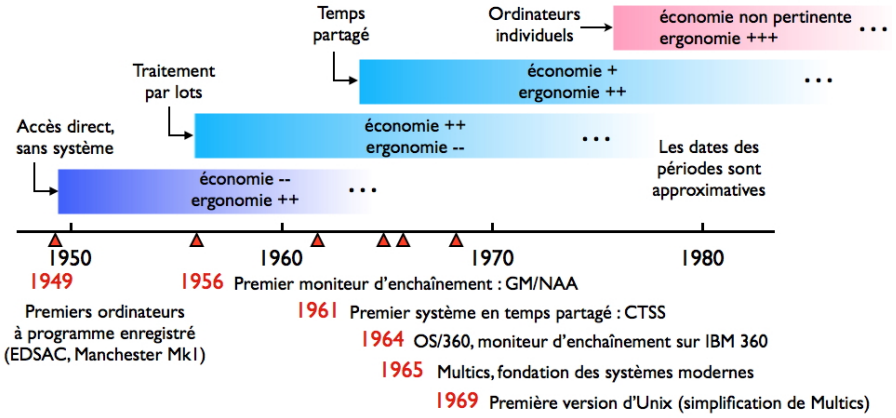

FIGURE 2. Chronologie simplifiée des premiers systèmes

seule. Plus tard furent introduits les sous-programmes, conservés dans une bibliothèque sous forme de rubans ; ceux-ci devaient être physiquement copiés, avant exécution, sur le ruban contenant le programme principal. Les ordres initiaux devaient donc assurer ce qu'on appelle aujourd'hui l'édition de liens, c'est-à-dire fixer les adresses d'appel et de retour des sous-programmes pour garantir leur exécution correcte. D'autres langages primitifs utilisant des sous-programmes furent développés plus tard, notamment sur UNIVAC par Grace Hopper. Leur traduction en langage machine était cette fois à la charge d'un programme enregistré, mais l'exploitation de l'ordinateur restait sous contrôle manuel.

Compte tenu de l'important investissement que représentait à l'époque l'achat d'un ordinateur, il n'est pas étonnant que l'on ait recherché les moyens de rendre son exploitation plus efficace. Les programmes des utilisateurs furent ainsi regroupés en lots, ou fournées (en anglais *batch*), pour les traiter en série, sans transitions. Une nouvelle fonction apparut, celle d'opérateur, technicien chargé de préparer les lots d'entrée à partir des programmes fournis par les utilisateurs, de surveiller l'exécution des travaux, et de distribuer les résultats de sortie. L'enchaînement des travaux était réalisé par un programme appelé moniteur, ou système de traitement par lots, qui fut la première forme de système d'exploitation. L'économie prit alors le pas sur l'ergonomie, les usagers étant désormais privés de l'accès libre à la machine et de la mise au point interactive.

1.2. La course à l'efficacité

L'évolution des systèmes de traitement par lots est dominée par la recherche de

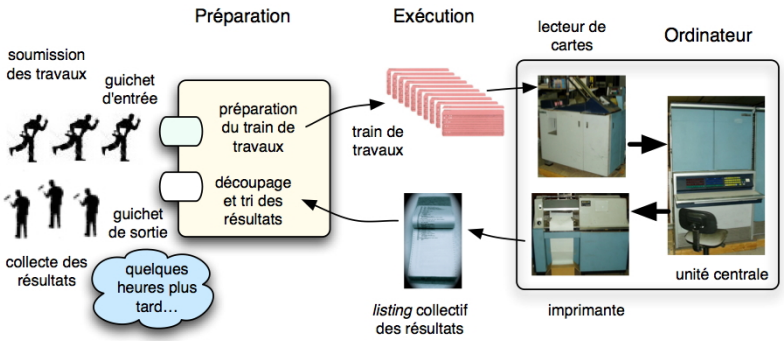

FIGURE 3. Organisation d'un système de traitement par lots

l'efficacité, au sens de l'utilisation optimale des ressources. Les innovations architecturales et matérielles (couplage de machines, multiprogrammation, canaux d'entrée-sortie) viseront cet objectif, sans grande considération pour le confort des utilisateurs.

Les principaux périphériques des ordinateurs des années 1950 étaient les lecteurs de cartes perforées, les imprimantes et les dérouleurs de bandes magnétiques. Dans le schéma initial, les travaux à exécuter avaient la forme d'un paquet de cartes, préparé par l'utilisateur ou par un atelier de perforation. Un programme résidant en permanence en mémoire, le moniteur d'enchaînement, avait pour rôle de lire les cartes, de lancer l'exécution des programmes et d'imprimer les résultats (Figure 3).

Il est intéressant de noter que le premier système d'exploitation fonctionnant sur ce principe, GM-NAA I/O, a été développé sur IBM 704 par des entreprises utilisatrices, General Motors et North American Aviation, et non par IBM, le constructeur de la machine. IBM devait fournir plus tard un système comparable, IBSYS, pour les ordinateurs de la série 7000.

Aussi simple soit-il, un tel système pose deux problèmes de sécurité.

— Si un programme d'utilisateur tourne en boucle infinie, par suite d'une faute de programmation, l'exécution du lot est bloquée indéfiniment (l'opérateur peut certes intervenir, mais il doit alors avoir une idée de la durée maximale du travail).

— Toujours à la suite d'une faute de programmation, un programme d'utilisateur peut écrire « n'importe où » dans la mémoire, et en particulier dans la zone réservée au moniteur. Il y a de nouveau un risque de blocage ou de dysfonctionnement de l'ensemble.

Deux innovations matérielles permirent de remédier à ces problèmes.

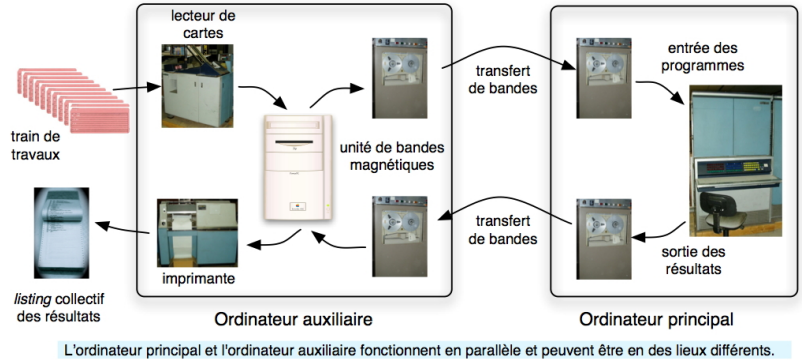

FIGURE 4. Traitement par lots avec ordinateur auxiliaire

En premier lieu, une horloge programmable, pouvant interrompre l'unité centrale après écoulement d'un délai spécifié. L'utilisateur devait alors préciser sur une carte de commande une durée maximale pour son travail, dans des limites fixées. Si ce temps d'exécution était dépassé, le moniteur interrompait le travail et passait au travail suivant. En second lieu, un dispositif permettant de protéger une zone de mémoire contre l'écriture. Le programme du moniteur était placé dans une zone ainsi protégée.

Cette organisation de système souffre d'un défaut d'efficacité, dû à la différence de vitesse entre les périphériques électromécaniques et l'unité de calcul. Pendant le temps de la lecture d'une carte ou de l'impression d'une ligne, l'unité centrale était oisive, car les ordinateurs de l'époque ne pouvaient faire qu'une chose à la fois.

Pour augmenter le taux d'utilisation de l'unité centrale, on eut l'idée de remplacer le lecteur de cartes et l'imprimante par des bandes magnétiques, organes beaucoup plus rapides. Un calculateur, auxiliaire, beaucoup plus simple que le calculateur principal, est utilisé pour constituer une bande d'entrée à partir des cartes soumises par les utilisateurs et pour imprimer le contenu de la bande de sortie. La fonction d'opérateur évolue pour prendre en compte ces changements. La figure 4 schématise ce mode d'exploitation (la phase de préparation est inchangée par rapport à la figure 3). Notons que la recherche de l'efficacité conduit à augmenter le nombre de travaux dans un lot, ce qui accroît encore le temps d'attente des utilisateurs.

Une association classique à cette époque de quasi monopole d'IBM consistait à utiliser un ordinateur 1401 pour traiter les bandes et un ordinateur de la série 7000, par exemple un 7094, pour les calculs.

Il restait encore un goulot d'étranglement : l'exécution séquentielle des programmes. Pendant la lecture d'un programme, l'unité centrale reste entièrement

occupée à cette opération, au détriment d'une activité plus productive. Deux progrès du matériel permirent de lever cette contrainte.

— L'invention des canaux d'entrée-sortie, processeurs autonomes capables de transférer des données depuis ou vers la mémoire en parallèle avec l'exécution de programmes par l'unité centrale.

— La mise en service de disques magnétiques, support de mémoire secondaire, avec une taille importante et un débit élevé de transfert. Dès lors, toute l'information utile, programmes et données, résidait en permanence dans cette mémoire secondaire.

Ces avancées, couplées à une augmentation de la taille de la mémoire, rendirent possible un nouveau schéma d'exécution, la multiprogrammation. Plusieurs programmes coexistent en mémoire centrale, ce qui permet de passer très rapidement de l'un à l'autre. De manière à occuper l'unité centrale à plein temps, les transferts depuis ou vers la mémoire secondaire se font en parallèle avec l'exécution des programmes. Ceux-ci sont dynamiquement implantés en mémoire en fonction de la place disponible, ce qui peut amener un programme à être chargé à des adresses différentes au cours de son exécution. Mais il faut alors résoudre le problème des adresses internes aux programmes, qui doivent être mises à jour quand un programme change de place. Ce problème trouva une solution élégante : les adresses ne sont plus absolues, mais relatives au début du programme, dont l'adresse est chargée dans un registre de base. Quand un programme est déplacé en mémoire, il faut seulement mettre à jour le contenu du registre de base qu'il utilise.

Au début des années 1960, comme expliqué dans [Rosin, 1969], on disposait donc de systèmes d'exploitation capables d'utiliser de manière quasi-optimale les capacités des gros ordinateurs de l'époque (*mainframes*). Un système emblématique de cette période est l'OS/360 d'IBM, dont la complexité était à la limite des capacités courantes de gestion du logiciel (voir à ce propos [Brooks, 1995]). Mais la recherche de performances économiques laissait largement de côté les besoins des utilisateurs, et notamment l'interactivité.

1.3. La convivialité perdue…

Lorsque l'ordinateur fonctionnait sans système d'exploitation sous le contrôle direct de l'utilisateur, ce dernier pouvait intervenir pendant l'exécution de ses programmes. Il pouvait par exemple arrêter l'exécution pour visualiser ou modifier au pupitre le contenu de cases de mémoire ; il avait en général la possibilité d'exécuter son programme en mode pas à pas, c'est-à-dire une instruction à la fois. En bref, il disposait d'une ébauche d'outils d'aide à la mise au point de ses programmes, mais les ressources de l'ordinateur étaient largement sous-utilisées.

Avec le traitement par lots, l'exploitation devient de plus en plus efficace, mais l'utilisateur a perdu tout contrôle sur l'exécution de ses programmes. Il se contente de déposer son travail au guichet du centre de calcul pour y récupérer les résultats

quelques heures plus tard. Et tant pis pour lui si une erreur typographique mineure a arrêté la compilation : il va devoir corriger et recommencer. La hausse de l'efficacité étant, de ce fait, pour une part illusoire, on s'est attaché à améliorer le confort (et donc la productivité) de l'utilisateur. Dans le contexte d'ordinateurs relativement puissants, mais en petit nombre, la solution passait par le partage d'un ordinateur entre une communauté d'utilisateurs simultanés. Le temps partagé allait naître.

1.4. Les débuts du temps partagé

Dès 1961, on trouvait sous la plume de John McCarthy, le créateur du langage LISP et le fondateur de l'intelligence artificielle, ces lignes prophétiques : *"[...] computing may someday be organized as a public utility just as the telephone system is a public utility"* (« [...] le calcul pourrait un jour être organisé comme un service public, exactement comme l'est le réseau téléphonique »). Au MIT (*Massachusetts Institute of Technology*), au début des années 1960, Fernando Corbató lança un projet inspiré par cette idée, CTSS (*Compatible Time-Sharing System*), décrit plus loin.

1.4.1. Les principes de base

Le principe du temps partagé est simple : il s'agit de tirer parti des différences de vitesse entre un ordinateur, cadencé alors à la microseconde, et un humain dont le temps d'interaction incluant réflexion et frappe de commande est de plusieurs secondes. L'ordinateur fonctionne alors comme un grand maître d'échecs jouant des parties simultanées. Lorsqu'il a traité la requête d'un utilisateur, il passe à celle de l'utilisateur suivant, puis à un autre et ainsi de suite. Si les requêtes soumises demandent un temps de traitement court par rapport au temps humain, un fonctionnement équilibré est possible avec un temps de réponse acceptable, même avec des centaines voire des milliers d'utilisateurs (voir Figure 5).

Si le principe est simple, la mise en œuvre est délicate. La multiprogrammation permet bien d'avoir plusieurs programmes en mémoire, mais pas des dizaines dans des mémoires de quelques centaines de kilooctets. Des techniques matérielles ont été introduites pour ne charger en mémoire que les parties utiles des programmes (la pagination) ou pour partager des programmes entre plusieurs utilisateurs avec une copie unique en mémoire (la segmentation). Chaque utilisateur possède ainsi une mémoire virtuelle (voir 2.2.2), qui lui donne l'illusion de disposer seul d'un grand espace adressable, isolé de celui des autres utilisateurs.

La conservation des programmes et des données de nombreux utilisateurs impose de disposer de systèmes de gestion de fichiers sur disque et d'utilitaires permettant de constituer ou modifier ces fichiers, les éditeurs.

Des outils interactifs de mise au point de programme redonnaient à l'utilisateur les mêmes possibilités que lorsqu'il était seul au pupitre de l'ordinateur. L'utilisateur pouvait déclencher des travaux par la frappe de commandes élémentaires (construire un fichier, compiler un programme contenu dans un fichier donné, exécuter le contenu d'un fichier, etc.) mais aussi combiner ces commandes dans un vrai

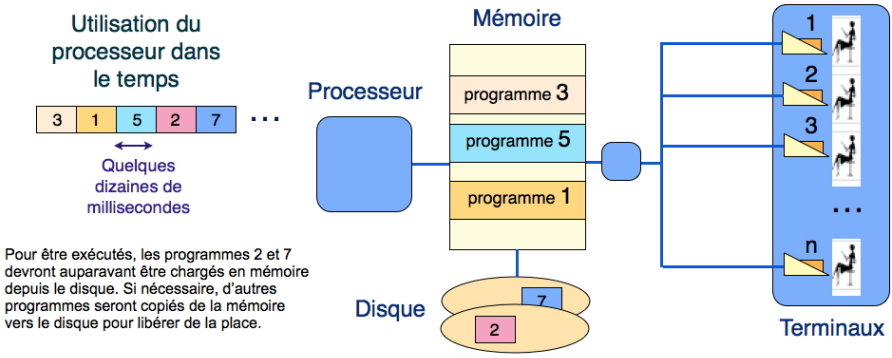

FIGURE 5. Architecture d'un système en temps partagé

langage de programmation (*shellscript*) comprenant en particulier des procédures avec des paramètres. On est loin du jeu de commandes rudimentaire des premiers systèmes de traitement par lots.

1.4.2. Les précurseurs

Les premiers systèmes en temps partagé furent développés au début des années 1960. Le pionnier fut CTSS [Corbató et al., 1962], qui fonctionnait en 1961 au MIT sur un ordinateur IBM 7094, modifié par IBM à la demande du MIT : taille de mémoire doublée, horloge programmable, dispositif de « trappe » (déroutement programmé lors de l'exécution de certaines instructions). CTSS était compatible avec le moniteur standard du 7094, qui pouvait s'exécuter en travail de fond (d'où le nom de *Compatible Time-Sharing System*).

En novembre 1961, lors de sa première démonstration, CTSS pouvait servir 4 utilisateurs, connectés via des terminaux mécaniques Flexowriter. Il fut mis en service effectif en 1963 et servait alors une trentaine d'utilisateurs, dont certains à distance, connectés par des lignes téléphoniques.

Les utilisateurs de CTSS disposaient de quelques outils, nouveaux pour l'époque, qui devaient devenir standard dans les systèmes futurs : éditeur de texte interactif, langage de commande évolué, exécutable au clavier ou à partir d'un fichier, messagerie permettant la communication entre utilisateurs, outil de formatage de texte pour l'impression.

Le principal apport de CTSS fut de montrer que l'idée du temps partagé était réalisable. Le MIT entreprit en 1963-64 un projet de plus grande ampleur, le projet MAC, qui devait donner naissance à Multics (voir 2.3.1).

Citons, sans être exhaustifs, quelques autres projets notables du début des années 1960.

— Le projet Genie fut mené à Berkeley sur un ordinateur SDS 930 de 1963 à 1967. Après une tentative manquée d'industrialisation des résultats de Genie, la plupart des membres de l'équipe (dont Butler Lampson et Chuck Thacker) rejoignirent le centre Xerox PARC où ils contribuèrent à la naissance de l'ordinateur personnel moderne [4].

— Le Dartmouth Time-Sharing System (DTSS) fut réalisé en 1963-64, initialement sur un General Electric GE-200. Il est connu pour avoir servi de support à BASIC, un langage de programmation simple spécialement dédié à un usage conversationnel. En 1970, le système, porté sur un GE-635, servait jusqu'à 300 terminaux.

— John McCarthy, qui avait quitté le MIT pour Stanford, réalisa en 1964-65 un système expérimental en temps partagé, Thor, qui fut le premier à utiliser des terminaux à écran.

Ces projets ont permis d'acquérir une expérience sur la construction de systèmes en temps partagé, préparant ainsi la voie à une transition du domaine des systèmes d'exploitation vers une véritable discipline scientifique fondée sur des principes rigoureux.

2. Naissance d'une discipline scientifique

Rappelons que le système d'exploitation d'un ordinateur a deux fonctions complémentaires : fournir aux utilisateurs une interface plus commode que celle de la machine physique pour réaliser et exploiter leurs applications ; gérer les ressources de cette machine pour les allouer aux activités qui en ont besoin. L'interface présentée aux utilisateurs par un système d'exploitation est celle d'une machine virtuelle, image idéalisée de la machine physique sous-jacente. La notion de virtualisation est centrale pour comprendre le fonctionnement des systèmes, et c'est d'abord à partir de cette notion qu'est née, dans les années 1965-70, l'approche scientifique des systèmes d'exploitation.

2.1. La virtualisation, clé des systèmes d'exploitation

En informatique, la virtualisation est une opération qui met en correspondance un objet concret (physique) et un objet abstrait (virtuel), image idéalisée du premier. C'est d'abord un outil intellectuel, qui facilite la conception d'un système informatique ; et l'association virtuel-physique trouve aussi une mise en œuvre concrète dans la phase de réalisation.

La virtualisation peut s'exercer de deux manières.

4. Voir l'article sur Interstices : « Xerox PARC et la naissance de l'informatique contemporaine », https://interstices.info/jcms/int_64091/xerox-parc-et-la-naissance-de-l-informatique-contemporaine.

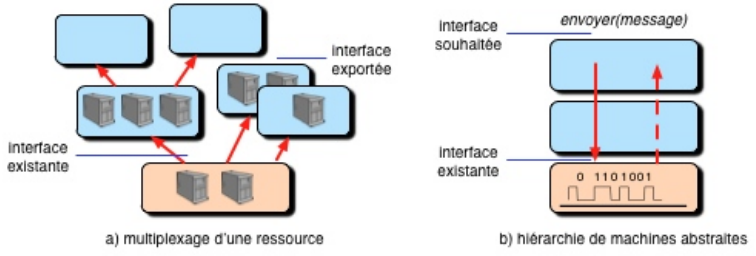

FIGURE 6. Les deux faces de la virtualisation

— Dans le sens ascendant (Figure 6a). On dispose d'une ressource physique, que l'on distribue à un ensemble d'utilisateurs par multiplexage (attribution de la ressource aux demandeurs par tranches de temps successives), créant ainsi des ressources virtuelles. C'est ainsi que l'on parle de processeur virtuel, de mémoire virtuelle, de machine virtuelle, notions examinées en détail dans la suite. L'interface exportée (celle des ressources virtuelles) peut être identique à celle de la ressource physique, ou peut en être différente. Une ressource virtuelle peut elle-même être virtualisée (c'est-à-dire partagée).

— Dans le sens descendant (Figure 6b). On souhaite réaliser un composant fournissant une certaine interface (dans l'exemple illustré, un envoi de message). On dispose d'un composant fournissant une interface différente (dans l'exemple, une liaison physique de bas niveau). Pour fournir l'interface souhaitée, on définit une hiérarchie descendante de composants (qu'on appelle machines abstraites), chacune utilisant la suivante, et la dernière s'appuyant sur l'interface existante. La conception procède ainsi par raffinements successifs, avec des va-et-vient éventuels.

La virtualisation s'est révélée un outil précieux pour la compréhension et la construction des systèmes d'exploitation, dans lesquels le partage des ressources par multiplexage est un mécanisme central. En outre, la réalisation d'un système fait souvent appel à la décomposition hiérarchique qui permet de traiter indépendamment ses différentes fonctions, qui relèvent elles-mêmes de la virtualisation. La première illustration de cette démarche a été apportée par Edsger Dijkstra dans un article [Dijkstra, 1968] resté célèbre.

2.2. Élaboration des concepts de base

Comment la notion de virtualisation s'est-elle concrétisée pour les principales ressources gérées par un système d'exploitation ? C'est ce que nous examinons maintenant, en considérant successivement le processeur, la mémoire et une machine tout entière.

2.2.1. Virtualisation du processeur

Le premier objet virtualisé a été le processeur (on parlait à l'époque d'unité centrale), organe d'exécution de la suite des instructions qui constituent un programme. L'intérêt de la virtualisation vient du fait que le processeur est utilisé pour exécuter plusieurs tâches. Contrairement à ce qu'on pourrait croire, c'est toujours le cas, même quand l'ordinateur n'a qu'un utilisateur unique. En effet, cet utilisateur fait souvent fonctionner plusieurs applications en parallèle (consulter sa messagerie, éditer un document, chercher sur le web, etc.). Et même s'il n'exécute qu'une seule application, le processeur doit gérer les communications (échanges sur le réseau, transferts vers le disque, gestion de l'écran et du clavier, etc.), même si l'essentiel du travail est délégué à des organes auxiliaires.

Le passage du processeur d'une tâche vers une autre se fait par un mécanisme appelé interruption : un signal est envoyé au processeur, qui suspend l'exécution de la tâche en cours, sauvegarde les données nécessaires à sa reprise ultérieure, reprend les données relatives à la nouvelle tâche et entame l'exécution de celle-ci. En l'absence de précautions particulières, cette interruption peut survenir à un moment quelconque, car elle est déclenchée par une cause extérieure à la tâche en cours : celle-ci peut être interrompue en n'importe quel point de son exécution. Les causes d'interruption peuvent être, par exemple, un signal d'horloge marquant la fin d'un temps d'exécution fixé, ou un signal de fin de transfert sur un disque, etc. Ainsi, si plusieurs tâches utilisent un processeur unique, ce dernier exécutera alternativement des séquences d'instructions de ces tâches. Celles-ci sont logiquement parallèles, au sens où elles sont simultanément en cours d'exécution. Mais sur le plan physique, le processeur n'exécute à un instant donné qu'une seule de ces tâches, les autres étant en attente : on parle alors de pseudo-parallélisme. Pour avoir un parallélisme réel, il faudrait disposer d'autant de processeurs que de tâches.

Utilisé sans autres précautions, ce mode de fonctionnement présente un risque potentiel. En effet, l'exécution d'activités parallèles manipulant des données communes peut rendre ces données incohérentes. Ce problème était initialement mal compris : dans les premiers systèmes d'exploitation, les entrées-sorties engendraient des dizaines d'interruptions par seconde, et la probabilité d'aboutir à des erreurs n'était pas négligeable. On attribuait souvent ces erreurs à des défaillances transitoires du matériel, mais il s'agissait bien de fautes de synchronisation.

En 1965, Edsger Dijkstra publie l'article fondateur de la théorie de la synchronisation [Dijkstra, 1965], en introduisant la notion de processus, activité séquentielle autonome pouvant interagir avec d'autres processus. Il formalise ainsi la notion intuitive de « tâche », travail confié à un ordinateur, en séparant la description du travail à effectuer (le programme), de l'entité dynamique, évolutive dans le temps, que constitue son exécution (le processus). C'est la différence entre la description d'une recette dans un livre de cuisine et son exécution concrète, avec l'usage d'ingrédients et d'ustensiles divers.

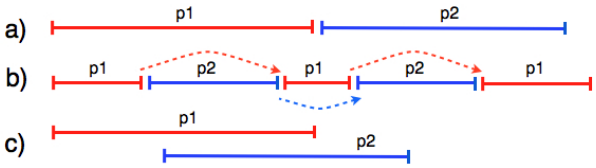

FIGURE 7. Modalités d'exécution de processus

On peut dès lors séparer l'aspect fonctionnel du processus (la description de ce qu'il fait) de la gestion des ressources nécessaires à son exécution : de ce point de vue fonctionnel, le processus peut être vu comme s'exécutant sur un processeur virtuel, toujours disponible. On dissocie ainsi l'activité logique, seule intéressante pour l'utilisateur, des contingences matérielles de son exécution (et notamment de l'allocation d'un processeur réel). Ces derniers aspects ne disparaissent pas, mais ils sont du ressort du système d'exploitation et restent invisibles à l'utilisateur. Ainsi, sur la figure 7, on voit trois modes d'exécution (séquentiel, pseudo-parallèle, et parallèle) de deux processus *p1* et *p2* ; mais d'un point de vue logique, les différences entre ces trois schémas d'exécution sont invisibles. Ne restent alors à régler que les interactions voulues entre processus, telles que l'échange d'information ou la création de sous-processus pour décomposer un travail en unités élémentaires.

Qu'il s'agisse de compétition entre processus (pour l'allocation de ressources, au niveau du système d'exploitation) ou de coopération (au niveau logique), il faut identifier et régler les cas d'incohérence potentielle, dont un exemple est la modification concurrente, par deux processus, de données communes. Dans cet exemple, la solution consiste à rendre ininterruptibles (on dit aussi « atomiques », car non sécables) les deux séquences d'instructions par lesquelles les deux processus modifient les données. Ainsi, leur effet global ne peut être que celui résultant de l'exécution de ces séquences l'une après l'autre, tout entrelacement étant exclu.

Les problèmes de synchronisation entre processus peuvent être réduits à un petit nombre de situations simples, dont celle illustrée ci-dessus, appelée exclusion mutuelle. Ces problèmes ont été élégamment résolus par divers mécanismes, dont les sémaphores, inventés par Dijkstra en 1965 [Dijkstra, 1965], et les moniteurs, à un niveau plus élevé d'expression, introduits au début des années 1970 par C. A. R. Hoare et Per Brinch Hansen [Hoare, 1974]. Plus tard, la notion d'exécution atomique a été étendue pour couvrir les cas de défaillance, grâce à la notion de transaction, due à Jim Gray, maintenant standard dans tous les systèmes de gestion de bases de données. Ces problèmes de synchronisation connaissent depuis quelques années un regain d'actualité avec l'apparition des processeurs multi-cœurs qui introduisent le parallélisme à un niveau beaucoup plus fin que dans les situations antérieures.

L'allocation de ressources (processeur, mémoire) aux processus peut donner lieu à deux phénomènes indésirables, qui ont été identifiés et compris à la fin des années 1960.

— L'interblocage (en anglais *deadlock*), qui se traduit par le blocage mutuel de plusieurs processus, dont chacun attend des ressources occupées par les autres.

— La privation, qui résulte de l'usage de priorités : un processus de faible priorité peut voir ses demandes indéfiniment retardées par une arrivée continue de processus plus prioritaires.

Divers remèdes sont connus, dont les plus usuels reposent sur l'utilisation d'un ordre : l'interblocage est évité si tous les processus demandent leurs ressources dans le même ordre ; la privation est évitée si les processus sont servis dans l'ordre de leur arrivée ; si on veut néanmoins utiliser des priorités, on peut faire croître la priorité d'un processus avec son temps d'attente.

2.2.2. Virtualisation de la mémoire

Dans les premiers ordinateurs, la mémoire centrale était une ressource coûteuse, donc peu abondante. Même dans les applications mono-utilisateur, elle était souvent trop petite pour contenir l'ensemble des programmes et des données. Les programmeurs devaient alors recourir à des schémas d'exécution avec recouvrement (*overlay*), où un découpage ingénieux des programmes permettait de les charger par « tranches » à la demande depuis une mémoire secondaire de grande capacité (tambour ou bande magnétique). Cela causait un surcroît de travail non négligeable pour les programmeurs et pénalisait le temps d'exécution.

Une avancée décisive fut réalisée en 1961 à l'université de Manchester, avec le concept de « mémoire à un niveau », mis en œuvre dans le système Atlas [Kilburn et al., 1961]. Les programmes d'application pouvaient utiliser un espace d'adressage beaucoup plus grand que celui disponible sur la mémoire principale, l'information étant contenue dans l'ensemble constitué par la mémoire principale et la mémoire secondaire à tambour. Deux notions essentielles étaient ainsi introduites :

— La dissociation entre les adresses utilisées par les programmes (nous les appelons « virtuelles », bien que ce terme n'ait été introduit que plus tard) et les adresses physiques des emplacements correspondants en mémoire principale ou secondaire.

— La gestion automatique (par le système d'exploitation, et non par les programmeurs) de la correspondance entre adresses virtuelles et adresses physiques.

Ce dernier aspect impliquait la gestion des transferts entre mémoire principale et mémoire secondaire pour amener en mémoire principale l'information nécessaire, lors de son utilisation. Ce transfert était réalisé par « pages » de taille fixe (sur l'Atlas, 512 mots). Comme les programmes étaient souvent réutilisés, un algorithme

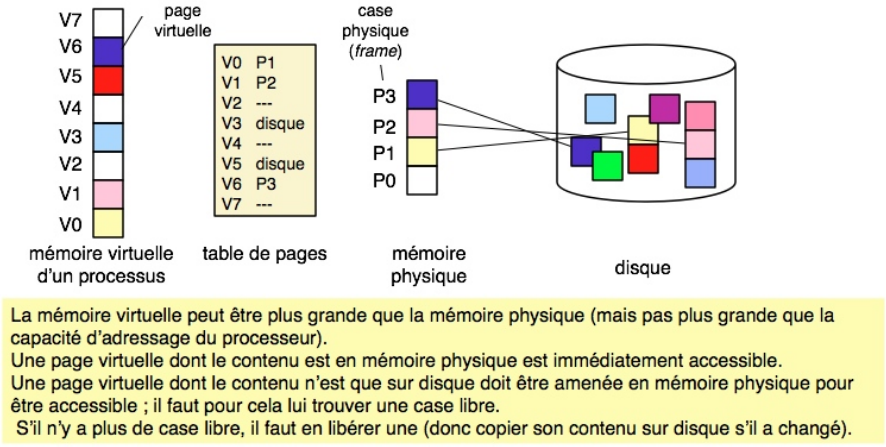

FIGURE 8. Réalisation d'une mémoire virtuelle

d'apprentissage était mis en place pour réduire le temps global de ces transferts (ses performances étaient voisines de celles obtenues par un programmeur expérimenté). Néanmoins, les performances se dégradaient rapidement quand la charge dépassait un certain seuil. Ce phénomène d'« écroulement » ne fut compris et traité qu'en 1968 (voir plus loin).

Dans les années 1965-68 furent posées les bases scientifiques de la mémoire virtuelle [Denning, 1970], toujours largement en vigueur aujourd'hui. Deux aspects complémentaires furent ainsi approfondis.

— L'organisation et la protection de la mémoire virtuelle, telle que vue par les utilisateurs.
— La mise en œuvre concrète de cette mémoire, notamment dans les systèmes en temps partagé, qui se développaient rapidement à cette époque.

La figure 8 montre schématiquement le principe de réalisation d'une mémoire virtuelle pour un processus. On a distingué, dans la mémoire physique, la notion de page (contenu) de la notion de case (contenant, ou suite d'emplacements de mémoire physique pouvant accueillir une page).

Dans ce schéma, la mémoire virtuelle est implicitement définie comme un espace unique d'un seul tenant. Un schéma plus élaboré [Dennis, 1965] fut proposé en 1965 par Jack Dennis, du MIT. Il repose sur la notion de segment, unité logique d'information, constitué d'une tranche de mémoire de taille variable composée d'emplacements contigus. Un utilisateur peut ainsi organiser ses informations en plusieurs segments (modules de programmes, différentes données, etc.). Des segments peuvent

être partagés entre plusieurs utilisateurs. L'accès aux segments doit pouvoir être protégé contre des accès non autorisés. Le schéma initial (illustré par la figure 8) devient un cas particulier, celui d'un segment unique de taille fixe.

Avec les performances des processeurs de l'époque, une mise en œuvre efficace de la segmentation supposait qu'elle soit inscrite dans l'architecture même de la machine. Quelques machines à mémoire segmentée furent ainsi réalisées, notamment le GE 645, utilisé par le système Multics décrit en 2.3.1. Néanmoins, l'idée d'une architecture à mémoire segmentée ne s'imposa pas généralement. Même si les processeurs actuels comportent une forme de support matériel pour la segmentation, ce mécanisme reste peu utilisé.

La notion de segment, gardant toutefois tout son intérêt, est simulée par une voie logicielle, en définissant des tranches de mémoire à l'intérieur d'une mémoire virtuelle unique de grande taille, elle-même réalisée par un mécanisme matériel de pagination. Les systèmes usuels comme Unix ou Windows utilisent cette solution et assurent, également par voie logicielle, la protection sélective des segments.

Dans un système multi-utilisateurs, comme l'étaient les systèmes en temps partagé du début des années 1970, chaque utilisateur peut exécuter un ou plusieurs processus, dont chacun dispose de sa mémoire virtuelle. L'isolation entre mémoires virtuelles est assurée par les tables de pages séparées, conservées par le système d'exploitation dans des zones protégées.

Bien que la mémoire virtuelle soit un outil de travail remarquable, il ne faut pas oublier qu'elle représente en quelque sorte un achat à crédit : pour que son contenu puisse être utilisé, un emplacement de mémoire virtuelle doit recevoir en fin de compte un support réel en mémoire centrale. Cet aspect avait été mal perçu par les concepteurs des premiers systèmes en temps partagé : à force de distribuer abondamment le « crédit », arrivait un moment où les ressources physiques ne pouvaient plus faire face aux demandes cumulées. C'était le phénomène de l'écroulement, qui conduisait à une paralysie complète du système.

Les causes de ce phénomène furent expliquées en 1968 en utilisant un modèle simple de comportement des programmes, et un remède fut trouvé, fondé sur l'observation et la limitation de la charge instantanée (de même qu'on traite la congestion d'une autoroute par une restriction temporaire des entrées). Le principe de la limitation de charge est toujours utilisé aujourd'hui, bien que la grande taille des mémoires physiques rende le problème beaucoup moins aigu.

2.2.3. Machines virtuelles

Après le processeur et la mémoire, l'étape suivante de la virtualisation est celle d'une machine tout entière. En fait, le terme de machine virtuelle recouvre plusieurs modes de fonctionnement. Nous nous intéressons seulement ici à la virtualisation d'une machine complète ; une autre forme de virtualisation, dont nous ne parlons pas, réalise un mécanisme d'interprétation des langages de programmation.

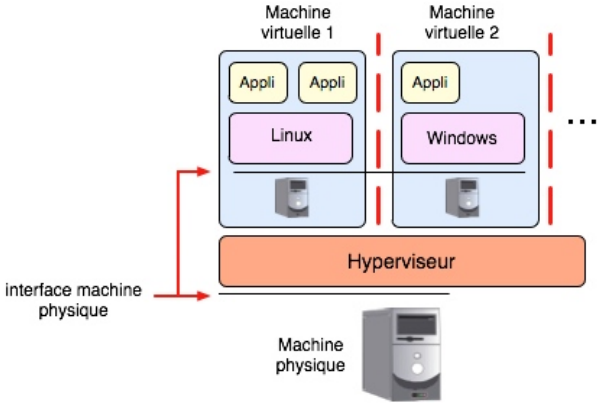

FIGURE 9. Machines virtuelles

Le principe d'un système de machines virtuelles, tel qu'il a été introduit par IBM entre 1965 et 1967, est schématisé sur la figure 9. Au-dessus d'une machine physique sont réalisées un certain nombre de machines virtuelles qui présentent à leurs utilisateurs une interface (jeu d'instructions, système d'interruptions, organes d'entrée-sortie) identique à celle de la machine sous-jacente. Ces machines sont complètement isolées les unes des autres : un programme s'exécutant sur une machine virtuelle ne peut lire ou modifier le contenu d'une autre machine virtuelle ; une défaillance logicielle d'une machine virtuelle n'a aucun effet sur les autres machines.

La réalisation d'un système de machines virtuelles repose sur un composant logiciel appelé hyperviseur, qui s'exécute directement sur la machine physique, dont il constitue en quelque sorte un « super système d'exploitation ». La réalisation d'un hyperviseur pose des problèmes techniques délicats car il doit virtualiser l'intégralité de la machine de base, y compris les mécanismes physiques de bas niveau tels que les interruptions, tout en assurant l'isolation mutuelle des machines virtuelles.

À quoi servent les machines virtuelles ? L'isolation mutuelle permet de faire fonctionner plusieurs systèmes d'exploitation différents sur une même machine physique. Néanmoins, la première utilisation des machines virtuelles a consisté à faire exécuter un grand nombre d'instances du même système, chacun servant un utilisateur unique : on simule ainsi un système en temps partagé. C'est le principe du système CP/CMS introduit en 1967 par le centre de recherche d'IBM de Cambridge sur l'ordinateur 360/67, spécialement conçu pour réaliser la mémoire virtuelle. Ce système, qui combinait un générateur de machines virtuelles (CP) et un logiciel interactif d'interprétation de commandes (CMS), exécuté sur chacune de ces machines, connut un grand succès, alors que le système d'exploitation « officiel » en temps partagé développé par IBM, TSS (*Time Sharing System*), ne réussit pas à atteindre ses objectifs.

Une extension du principe des machines virtuelles consiste à simuler des machines ayant un jeu d'instructions différent de celui de la machine physique sous-jacente. On peut ainsi expérimenter le jeu d'instructions d'une machine avant de la construire, ou faire exécuter des programmes écrits pour une machine dont on ne dispose pas d'exemplaire physique.

Dans les années 1980, le domaine des machines virtuelles connut un certain déclin, en raison du développement des ordinateurs personnels. Néanmoins, au milieu des années 1990, l'expansion des systèmes répartis et des multiprocesseurs conduisit à des configurations complexes, pour lesquelles les machines virtuelles étaient un bon outil de structuration. On assiste depuis lors à une renaissance des machines virtuelles [Smith and Nair, 2005], avec notamment la création d'entreprises spécialisées dans leur construction et leur mise en place.

2.3. De Multics à Unix

La conception du système Multics [Organick, 1972] marque la transition entre l'ère du savoir-faire et l'ère scientifique dans le domaine des systèmes d'exploitation. Bien que sa carrière commerciale ait été tardive et limitée, les avancées conceptuelles et techniques dont il a été le lieu font de ce projet un des éléments fondateurs de la discipline des systèmes d'exploitation. Multics a par ailleurs donné naissance à Unix, l'un des systèmes les plus répandus aujourd'hui.

2.3.1. Multics

Le projet MAC (*Man And Computer*) débuta au MIT en 1963-64, sur la lancée du succès de CTSS. C'était un projet de grande envergure, largement financé par l'ARPA (organisme de financement militaire de la recherche). L'objectif principal était de réaliser un système en temps partagé capable de servir une très large communauté d'utilisateurs. Ce système fut appelé Multics (*Multiplexed Information and Computing Service*).

L'expérience de CTSS avait montré la faisabilité du projet, mais aussi le rôle crucial de dispositifs matériels spécialement adaptés au fonctionnement en temps partagé. Aussi fut-il décidé d'associer au projet deux entreprises industrielles : General Electric, qui devait concevoir et réaliser la machine support, et les Bell Labs, qui devaient participer à la conception du logiciel et notamment aux aspects liés à l'accès à distance.

La réalisation de cet ambitieux projet se révéla plus difficile que prévu. Le système commença à fonctionner en 1969, sur l'ordinateur GE-645 conçu par General Electric, mais ses performances étaient alors très loin des objectifs fixés. Les Bell Labs se retirèrent du projet cette même année. En 1970, la division informatique de General Electric fut rachetée par Honeywell, qui poursuivit le développement de Multics comme produit commercial. Cette division informatique fut reprise par Bull en 1985 et Multics continua sa carrière dans cette entreprise. Le MIT poursuivit, pour sa part, la réalisation de Multics comme projet de recherche et support de services.

Multics a utilisé et perfectionné tous les concepts scientifiques de base des systèmes d'exploitation : processus et synchronisation, segmentation et liaison dynamique, mémoire virtuelle paginée, système de gestion de fichiers, sécurité et protection, structuration, etc. Il a été utilisé pendant plusieurs décennies, le dernier site Multics ayant été arrêté en 2000.

Du point de vue de l'utilisateur, Multics a mis en évidence l'importance de l'organisation structurée et du partage de l'information, via le système de gestion de fichiers. Le modèle de catalogues hiérarchiques, de protection sélective et de liens, établi par Multics, est toujours à la base des systèmes actuels.

2.3.2. Unix

Après le retrait des Bell Labs du projet Multics, quelques ingénieurs des Bell Labs ayant travaillé sur ce projet, emmenés par Ken Thompson et Dennis Ritchie, décidèrent de prendre le contrepied de ce système qu'ils jugeaient lourd et complexe, en utilisant leur expérience pour réaliser un système minimal sur une petite machine. Ayant accès à un DEC PDP-7 peu utilisé, ils commencèrent en 1969, pour leur propre compte et sans aucun soutien des Bell Labs, le développement d'un système conversationnel mono-utilisateur qu'ils baptisèrent Unics (jeu de mots sur Multics).

Devenu multi-utilisateurs, le système prit le nom d'Unix [Ritchie and Thompson, 1974], fut porté sur PDP-11/20 et commença en 1970 à être soutenu et utilisé par les Bell Labs. La première installation commerciale eut lieu en 1972. Cette même année, Unix fut réécrit dans un nouveau langage de programmation, C, créé par Dennis Ritchie. Ce langage, proche de la machine physique (il permet notamment de manipuler des adresses), présente des constructions de haut niveau et son compilateur produit un code efficace. Il devait devenir le langage privilégié pour l'écriture du logiciel de base.

Unix, devenu portable, commença alors à se diffuser rapidement dans les universités et centres de recherche, créant une communauté très active qui contribua à l'évolution du système. Diverses versions virent le jour, notamment chez plusieurs constructeurs de machines. Les tentatives d'unification n'aboutirent pas, et Unix, sous différentes variantes, est toujours en usage aujourd'hui.

Unix est organisé autour d'un noyau de base conçu pour être efficace, au détriment d'une structuration claire. Ce noyau sert de support à un interprète de langage de commande, le *shell*, dont il existe de nombreuses versions. Pour l'utilisateur, le langage de commande du *shell* permet de réaliser des tâches complexes par assemblage d'actions élémentaires, en séquence ou en parallèle. Le mécanisme des tubes (*pipes*) est particulièrement commode pour transmettre des informations entre des commandes successives. Le système de fichiers est dérivé de celui de Multics, avec un mécanisme de protection moins élaboré.

3. Les temps modernes

La période 1970-80 fut marquée par un développement foisonnant de systèmes d'exploitation. Mais à partir du milieu des années 1980, on assista à une stabilisation du domaine autour d'un petit nombre de systèmes, situation qui prévaut encore aujourd'hui. Les phénomènes marquants de la période 1975-2000 ont été d'abord la large diffusion des ordinateurs personnels, puis la montée en puissance des réseaux, devenus une extension obligée des ordinateurs. À partir des années 2000, l'apparition et la diffusion massive des appareils mobiles dominent le paysage, mais sans apporter de révolution dans le domaine des systèmes d'exploitation, dont les acquis s'adaptent aux situations nouvelles.

3.1. Les ordinateurs personnels

Les ordinateurs personnels se développèrent à partir de 1972, à la suite de la réalisation des premiers microprocesseurs. Il étaient initialement vendus en « kit », et leur public était surtout constitué d'amateurs versés en montages électroniques. La situation changea à partir de 1977 ; après l'Apple II, les ordinateurs tout montés devinrent prédominants. Ces premiers ordinateurs étaient équipés du système d'exploitation CP/M, distribué par la société Digital Research dirigée par Gary Kildall. CP/M était un système de conception simple et rigoureuse, organisé en couches et prévu pour un usager unique.

Vers la même époque (1973-74), le centre de recherche Xerox PARC réalisait un ordinateur personnel de conception révolutionnaire, l'Alto. Cette machine, destinée à organiser le « bureau du futur », privilégiait l'interaction avec l'utilisateur, via une interface graphique évoluée, ainsi que la communication, grâce à l'intégration dans un réseau local. Son système d'exploitation, l'Alto-OS [Lampson and Sproull, 1979], était conçu en fonction de ces objectifs, mais la diffusion de ce système, comme celle de l'Alto, restait limitée à Xerox.

En 1981, la société IBM, consciente de l'importance croissante de l'informatique individuelle, mettait sur le marché l'ordinateur PC. Contrairement à sa stratégie antérieure, IBM fit largement appel à la sous-traitance pour le projet PC. IBM souhaitait construire son système d'exploitation à partir de CP/M, qui avait démontré son efficacité, mais les négociations avec Gary Kildall n'aboutirent pas. Alors se présenta la société Microsoft, récemment créée par Bill Gates et Paul Allen, avec un système d'exploitation appelé QDOS (*Quick and Dirty Operating System*), lui-même inspiré par CP/M, racheté à une petite entreprise, Seattle Computer Products. QDOS, rebaptisé MS-DOS après des modifications mineures, fut adopté par IBM pour son PC, ce qui marqua le début de la fortune de Microsoft.

En parallèle, les idées novatrices mises en œuvre à Xerox PARC autour de l'Alto furent exploitées d'abord en 1981 par Xerox pour son Star, puis en 1983 par Apple pour son Lisa. Mais ces machines, bien que techniquement remarquables, n'eurent

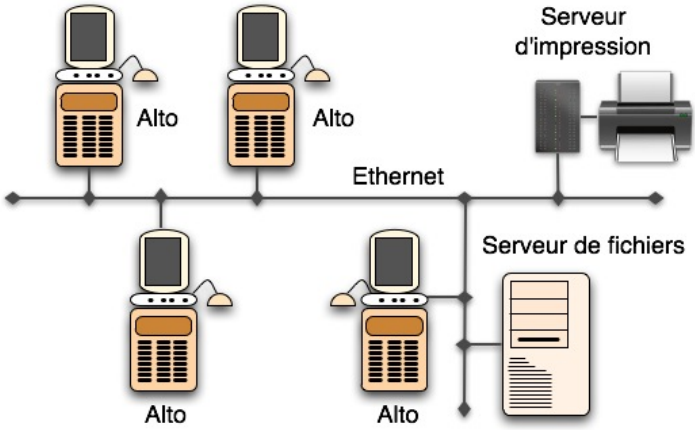

FIGURE 10. Le réseau local de Xerox PARC

pas de succès commercial en raison de leur coût élevé, des performances médiocres de leurs applications, et d'un marché encore réticent à la nouveauté. Ce n'est qu'avec le Macintosh, sorti par Apple en 1984, que furent diffusées dans le grand public comme dans les entreprises les innovations issues de Xerox PARC.

La situation des ordinateurs personnels évolua peu dans les 20 années suivantes, en dehors de l'amélioration des performances portée par la loi de Moore et les progrès de la technologie des disques. MacOS (successeur d'AltoOS), et Windows (successeur de MS-DOS, inspiré par l'interface graphique du Macintosh), évoluèrent en parallèle au cours de leurs versions successives. Les dernières versions de MacOS (MacOS X) s'appuient sur un substrat Unix lui-même réalisé sur un micro-noyau (voir 3.3) inspiré de Mach. Les systèmes d'exploitation pour les *smartphones* apparus au cours des années 2000 sont directement dérivés de systèmes existants (iOS, chez Apple, issu de MacOS X, et Android, chez Google, issu de Linux, une variante d'Unix).

3.2. Le réseau, complément obligé du système

En 1972-73, l'Arpanet (réseau à grande distance prédécesseur de l'Internet) faisait ses premiers pas, et servait surtout à des échanges entre ordinateurs distants, comme le transfert de fichiers, la connexion à distance ou le courrier électronique. À cette époque, les chercheurs de Xerox PARC développaient un autre modèle d'organisation répartie, où un ensemble de serveurs implantés sur un réseau local (en l'occurrence Ethernet) sert d'extension aux ordinateur connectés, en leur fournissant un ensemble de services tels que le stockage de fichiers ou l'impression (Figure 10).

Cette organisation se répercute sur le système d'exploitation, dans lequel les services locaux et les services distants jouent un rôle analogue. La communication avec les serveurs distants se fait par un protocole dit client-serveur, reposant sur un appel de procédure à distance qui étend l'appel de procédure ordinaire au cas où appelant et appelé sont sur des sites distincts.

Ce modèle sera notamment exploité par les stations de travail, apparues au début des années 1980, ordinateurs puissants souvent utilisés par un usager unique, et reposant largement sur des ressources accessibles par un réseau. Le slogan affiché par Sun Microsystems, *The network is the computer*, décrit bien cette situation. Dans le système d'exploitation, des instruments tels que les *sockets*, initialement développées pour Unix mais depuis largement étendues, simplifient la communication entre machines sur un réseau en fournissant une interface identique à celle d'un système de fichiers.

Du schéma client-serveur, on passa à un modèle plus général de système réparti, où un ensemble de machines connectées coopérantes fonctionne comme une machine unique, avec l'avantage d'un gain de performances et d'une forte résistance aux défaillances. Il fallut développer de nouveaux concepts pour organiser de tels systèmes, comme le consensus ou la validation atomique de transactions.

3.3. Informatique embarquée et micro-noyaux

Un système embarqué associe un organe matériel et un système logiciel, qui est incorporé à cet organe et commande son fonctionnement en vue d'une fonction spécifiée. Cette définition recouvre des situations très diverses : logiciel inclus dans un téléphone mobile, un appareil ménager, un processeur interne à un véhicule, un système de navigation, etc. Ces systèmes sont soumis à des contraintes fortes sur leur capacité de mémoire, leur puissance de traitement, leur consommation en énergie, la bande passante de leurs organes de communication. Leur cahier des charges impose souvent des exigences de disponibilité et de réactivité (échéances dures en temps réel). Les systèmes embarqués sont plongés dans un environnement physique changeant, ce qui leur impose des capacités d'adaptation.

Les systèmes embarqués connaissent un développement rapide, dû à l'intégration de l'informatique et des communications dans un grand nombre d'activités et de dispositifs du monde physique : l'informatique est dite « omniprésente » (*pervasive*). Un système d'exploitation pour systèmes embarqués doit s'adapter aux contraintes ci-dessus, et donc posséder les qualités d'économie, d'ouverture, et de capacité d'évolution

Ces exigences impliquent une redéfinition radicale de l'organisation d'un système d'exploitation. Deux voies sont suivies pour cela.

— Construire une version « réduite » d'un système d'exploitation classique. Les systèmes iOS et Android, cités plus haut, en sont des exemples.

— Réaliser un système « à la carte » à partir de la base, en utilisant des canevas (*framework*) à composants.

On assiste dans ce domaine à une renaissance des micro-noyaux. Introduite dans les années 1980, l'architecture à micro-noyau vise à concentrer les fonctions élémentaires de gestion du processeur, de la mémoire et des communications, dans une couche logicielle réduite, les autres composants (gérant de processus, de mémoire virtuelle, de fichiers, de réseaux, etc.) étant réalisés par des serveurs intercommunicants utilisant les fonctions du micro-noyau. Un système traditionnel tel qu'Unix peut alors être réalisé par un ensemble de serveurs. Un système spécialisé (gérant de communication, logiciel pour un téléphone portable) ne comporte que les serveurs strictement nécessaires à ses fonctions.

Les micro-noyaux n'ont pas initialement répondu à toutes les attentes qu'ils avaient suscitées, en raison de leurs performances insuffisantes et de la difficulté à modifier les politiques de base préinscrites dans les gérants de ressources. Ils ont connu un déclin relatif avant de revenir comme support de systèmes embarqués à base de composants, facilitant l'adaptation des systèmes à un environnement particulier. La concentration des fonctions de base dans un noyau réduit permet en outre d'isoler la « base de confiance » (partie du système critique pour la sécurité) à l'intérieur de ce noyau. Un objectif à long terme est la génération automatique de systèmes spécialisés à partir de spécifications déclaratives de leurs fonctions et de leur environnement.

3.4. Vers une normalisation de fait

Au cours des années 1970, un grand nombre de systèmes d'exploitation furent réalisés : citons IBM VM/370, DEC TOPS10 et Vax VMS, BBN Tenex. Mais depuis la fin des années 1980, si on exclut le domaine déjà mentionné de l'informatique embarquée et des objets connectés, on assiste à une concentration sur un très petit nombre de systèmes : Windows pour les PC, MacOS pour les Macintosh, diverses variantes d'Unix (dont le logiciel libre Linux), et z/OS pour les *mainframes* d'IBM. Ce phénomène s'explique, d'une part par la concentration des architectures de machines, d'autre part par le coût de développement d'un nouveau système d'exploitation, devenu très élevé.

4. Quelques défis actuels

La relative stabilité observée dans le domaine des systèmes d'exploitation ne doit pas faire oublier que de nombreux problèmes restent encore largement ouverts. Nous les résumons brièvement ci-après.

— ***Sécurité.*** La sécurité reste le problème majeur des systèmes d'exploitation, pour deux raisons principales : l'absence, pour le moment, d'un fondement théorique satisfaisant pour la sécurité ; et la complexité même des systèmes et

des architectures de machines et de réseaux, qui rend très difficile une analyse complète des voies d'attaque possibles.

— ***Certification.*** Certifier un système, c'est prouver qu'il répond à ses spécifications. Bien que cet objectif ait longtemps été considéré comme inaccessible pour un système d'exploitation, on peut noter des avancées récentes spectaculaires, comme la preuve rigoureuse de la validité d'un noyau de système [Klein et al., 2009].

— ***Systèmes embarqués et mobiles.*** Comme indiqué en 3.3, le domaine des systèmes embarqués et mobiles est en pleine expansion. Il présente des difficultés spécifiques comme le passage à très grande échelle, l'interaction avec un environnement physique complexe et les contraintes d'environnement, telles que la limitation de la consommation d'énergie.

— ***Parallélisme.*** Le problème de la gestion d'activités parallèles reste encore largement tributaire d'un traitement *ad hoc*, en l'absence d'une théorie complète de ce domaine. Ce problème est encore compliqué par le développement récent de processeurs multicœurs, qui introduit le parallélisme à un niveau beaucoup plus fin que par le passé.

— ***Virtualisation à grande échelle.*** L'informatique en nuage (*cloud computing*) est une forme de service dans lequel les fonctions de traitement et de stockage de l'information sont déléguées à une infrastructure virtuelle implantée sur un réseau de centres de données, dont chacun concentre un nombre élevé de serveurs. Cette organisation pose divers défis auxquels doivent répondre les systèmes d'exploitation des serveurs : maîtrise de la consommation d'énergie, garantie de la qualité de service, confidentialité.

— ***Autonomie et adaptation.*** Un système doit être maintenu en état de remplir ses fonctions malgré des événements indésirables et imprévus : défaillance, pic de charge, attaque. Les systèmes autonomes visent à rendre partiellement ou entièrement automatiques ces tâches d'administration, grâce à des méthodes issues de l'automatique (commande par boucles de rétroaction) et de la statistique (prévisions de charge fondées sur l'observation, détection de causes d'anomalies).

Références

Références générales.

Le livre en ligne [Bloch, 2003] fournit des indications historiques sur les divers aspects des systèmes d'exploitation. Les livres [Brinch Hansen, 2002] et [Brinch Hansen, 2010] reproduisent des articles historiques sur les débuts de la programmation concurrente et sur des systèmes d'exploitation novateurs. L'article [Rosin, 1969] est une référence pour l'histoire des premiers systèmes de traitement par lots.

[Bloch, 2003] Bloch, L. (2003). *Les systèmes d'exploitation des ordinateurs. Histoire, fonctionnement, enjeux*. Vuibert, Paris. Version en ligne : *http ://www.laurentbloch.org/Data/Livre-Systeme/*.

[Brinch Hansen, 2002] Brinch Hansen, P. (2002). *The Origin of Concurrent Programming : From Semaphores to Remote Procedure Calls*. Springer.

[Brinch Hansen, 2010] Brinch Hansen, P. (2010). *Classic Operating Systems : From Batch Processing to Distributed Systems*. Springer.

[Brooks, 1995] Brooks, Jr., F. P. (1995). *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Corbató et al., 1962] Corbató, F. J., Merwin-Daggett, M., and Daley, R. C. (1962). *An Experimental Time-sharing System*, pages 117–137. In [Brinch Hansen, 2010]. Publication initiale : *Spring Joint Computer Conf. 21*, pages 335–344, 1962.

[Denning, 1970] Denning, P. J. (1970). Virtual Memory. *ACM Comput. Surv.*, 2(3) :153–189.

[Dennis, 1965] Dennis, J. B. (1965). Segmentation and the Design of Multiprogrammed Computer Systems. *J. ACM*, 12(4) :589–602.

[Dijkstra, 1965] Dijkstra, E. W. (1965). *Cooperating Sequential Processes*, pages 65–138. In [Brinch Hansen, 2002]. Publication initiale : Technical Report EWD 123, Burroughs, Nuenen (Sept. 1965) ; repris dans : *Programming Languages*, F. Genuys, Ed., Academic Press, New York, 1968, pages 43-112.

[Dijkstra, 1968] Dijkstra, E. W. (1968). *The Structure of the THE Multiprogramming System*, pages 223–236. In [Brinch Hansen, 2010]. Publication initiale : *Commun. ACM*, 11 :5, pages 341–346, May 1968.

[Hoare, 1974] Hoare, C. A. R. (1974). *Monitors : An Operating System Structuring Concept*, pages 272–294. In [Brinch Hansen, 2002]. Publication initiale : *Commun. ACM*, 17 :10, pages 549-557, October 1974.

[Kilburn et al., 1961] Kilburn, T., Payne, R. B., and Howarth, D. J. (1961). *The Atlas Supervisor*. In [Brinch Hansen, 2010]. Publication initiale : *AFIPS Computer Conf. 20*, pages 279–294, 1961.

[Klein et al., 2009] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. (2009). seL4 : Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP'09, pages 207–220, New York, NY, USA. ACM.

[Lampson and Sproull, 1979] Lampson, B. W. and Sproull, R. F. (1979). *An Open Operating System for a Single-user Machine*, pages 414–432. In [Brinch Hansen, 2010]. Publication initiale : in *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP'79, pages 98–105, New York, NY, USA, 1979. ACM.

[Organick, 1972] Organick, E. I. (1972). *The Multics System : An Examination of Its Structure*. MIT Press, Cambridge, MA, USA.

[Rosin, 1969] Rosin, R. F. (1969). Supervisory and Monitor Systems. *ACM Comput. Surv.*, 1(1) :37–54.

[Smith and Nair, 2005] Smith, J. E. and Nair, R. (2005). *Virtual Machines : Versatile Platforms for Systems and Processes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.