La recherche sur les systèmes, des pivots dans l'histoire de l'informatique

Camille Paloque-Bergès, Loïc Petitgirard

le cnam

Cahiers d'histoire du Cnam

vol. 7–8

Dossier

La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II

Coordonné par Camille Paloque-Berges et Loïc Petitgirard

2017 / Second semestre
(nouvelle série)

Cahiers d'histoire du Cnam

vol. 7–8

Dossier

La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II

Coordonné par Camille Paloque-Berges et Loïc Petitgirard

2017 / Second semestre
(nouvelle série)

Cahiers d'histoire du Cnam. Vol. 7-8, 2017 / 2 (nouvelle série).
Dossier « **La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II** », coordonné par Camille Paloque-Berges et Loïc Petitgirard.

Secrétariat de rédaction
Camille Paloque-Berges, assistée de **Sofia Foughali**
Laboratoire HT2S-Cnam, Case 1LAB10,
2 rue Conté, 75 003 Paris
Téléphone : 0033 (0)1 40 27 22 74
Mél : camille. paloque_berges@cnam.fr
sofia. foughali_sadji@cnam.fr

Maquettage
Françoise Derenne, *sur un gabarit original créé par la Direction de la Communication du Cnam*

Impression
Service de la reprographie du Cnam

ISSN 1240-2745

Fondateurs (première série, 1992)
Claudine Fontanon, André Grelon

Ce double volume a été réalisé
grâce au soutien du **Laboratoire d'Excellence HASTEC**

haStec
Laboratoire d'Excellence
Histoire et anthropologie
des savoirs, des techniques
et des croyances

Les 5 premiers numéros de l'ancienne série (1992-1996) sont disponibles intégralement sur le site Web du Conservatoire numérique du Cnam : **http://cnum.cnam.fr**

Sommaire

Dossier

Dossier

La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II

Coordonné par Camille Paloque-Berges et Loïc Petitgirard

Introduction
Éléments d'histoire des systèmes d'exploitation des années 1960 aux années 1990

Camille Paloque-Berges
HT2S, Cnam.

Loïc Petitgirard
HT2S, Cnam.

Ce deuxième volume du double numéro consacré à la recherche sur les systèmes comme pivots dans l'histoire de l'informatique est le fruit de réflexions développées lors de deux colloques organisés conjointement : par l'équipe du séminaire « Histoire de l'informatique et du numérique » qui se déroule au Musée des arts et métiers depuis de nombreuses années, avec pour focalisation les objets, les périodes, et les lieux institutionnels et industriels de l'histoire de l'informatique, en particulier en France, et à partir du point de vue privilégié des acteurs techniques de cette histoire ; et par celle du programme « Hist.Pat.info.Cnam », qui a regroupé entre 2014 et 2017 sous l'égide du LabEx Histoire, Anthropologie des Savoirs, Techniques et Croyances (HASTEC) des chercheurs interdisciplinaires de plusieurs laboratoires (histoire, information-communication, informatique) pour réfléchir à la formation et à la légitimation institutionnelle de la discipline informatique ainsi qu'à la documentation de son histoire. Ce numéro élargit la perspective après une première focalisation sur le cas du Conservatoire national des arts et métiers. Ces travaux ont illustré la montée en recherche, à partir des années 1970, d'acteurs de l'informatique au Cnam ayant accompagné le projet scientifique de la création du laboratoire le Centre d'Études et de Recherches en Informatique du Cnam (Cédric) à la fin des années 1980[1].

L'histoire des systèmes d'exploitation est à la fois une problématique déjà posée et un terrain en friche au sein

1 Voir le premier volume de ce double numéro.

de l'historiographie de l'informatique. Celle-ci ne limite pas ses réflexions à l'histoire des machines mais s'intéresse à l'évolution des concepts, processus et applications logicielles ainsi qu'aux pratiques, manières de faire, circulation et diffusion des savoirs vers les usages. Cette histoire se penche sur les logiciels (*software*), dont l'importance est croissante depuis quinze ans pour les historiens dès lors qu'ils sont pleinement considérés comme un artefact non seulement technique, mais aussi industriel, commercial, voire culturel, reposant sur des efforts de développement considérables (Hashagen, 2002 ; Campbell-Kelly, 2003). En ce sens, toute la problématique du logiciel, de l'organisation des communautés de pratiques jusqu'à la question de la brevetabilité du logiciel, traverse l'histoire des systèmes d'exploitation.

Mais l'histoire des systèmes ne saurait se dissoudre dans l'histoire générale du logiciel, en ceci qu'elle suit une dynamique qui lui est propre. De fait, le système d'exploitation est un artefact central de la machine qui organise et distribue les processus, alloue un temps et un espace à l'utilisateur pour ses opérations, assure l'interface entre l'humain et la machine. Les années 1970 sont une décennie d'accélération dans la recherche et le développement de systèmes d'exploitation, transition entre les grands systèmes et mini-ordinateurs, d'usages industriel et scientifique, et la micro-informatique, d'usage général. Si ce numéro se concentre essentiellement sur ce moment de transition, il propose également une réflexion sur l'évolution de la notion et celle du domaine des systèmes en informatique en général. Par exemple, les systèmes de bases de données étaient, à leurs débuts, souvent considérés comme des éléments des systèmes d'exploitation, avant d'être définitivement placés dans la catégorie « applications », catégorie à part entière.

Durant les deux décennies précédentes, la recherche sur les grands systèmes a fait émerger d'abord des « systèmes de contrôle », avec une grande variété d'implémentations chez les constructeurs. Le texte de M. Bullynck propose une synthèse de ces développements sur la période 1954-1964, qui préparent les systèmes ultérieurs, leur dissémination et ce qui bientôt s'appellera *operating systems*. De manière complémentaire, C. Kaiser, ancien professeur titulaire de la chaire « Informatique-programmation » au Cnam[2], offre le point de vue d'un acteur de cette histoire des systèmes, et montre dans son texte le contexte technique, industriel et académique dans lequel la question des systèmes d'exploitation devient une discipline à part entière à la fin des années 1960.

Tirant parti des expériences des années 1960, les mini-ordinateurs seront

2 Claude Kaiser est un personnage central de l'histoire racontée dans le premier volume de ce double numéro, auquel nous renvoyons, ainsi qu'à un entretien publié dans la revue *TSI* (Neumann, Petitgirard, Paloque-Berges, 2016).

associés aux systèmes dits répartis, qui permettent de gérer des processus multi-utilisateurs en temps réel. L'ordinateur n'est plus un grand système qui n'est manipulable que par des utilisateurs très spécialisés, les opérateurs, mais rassemble autour de lui une équipe de spécialistes aux compétences diversifiées. Dans les années 1970, les laboratoires de R&D s'équipent systématiquement de nouvelles machines mini-informatiques moins onéreuses, plus puissantes, plus flexibles aussi bien pour des opérations de calcul scientifique, de programmation d'applications, que des usages plus généraux de traitement et transfert de données.

Le développement de langages de programmation (pour les systèmes et les applications) joue un rôle déterminant dans ces avancées, permettant de travailler à « haut niveau » sur la machine, c'est-à-dire à travers des langages symboliques plus proches du langage naturel, et non plus « à bas niveau » avec une programmation par câblages ou par langage-machine, proches du codage binaire. Ces développements génèreront également des expériences originales de machines informatiques cherchant à déplacer la frontière entre matériel et logiciel au profit du matériel, espérant gagner en performances. Dans ce dossier, un autre acteur et professeur au Cnam (ancien titulaire de la chaire de Physique des composants électroniques), F. Anceau, décrit précisément l'histoire des « machines-langages » (exécutant directement du langage évolué) et des « machines-systèmes » intégrant des mécanismes de systèmes dans leur matériel.

L'histoire des systèmes d'exploitation se matérialise dans les espaces de la recherche en informatique dans les laboratoires scientifiques des universités et des entreprises. Au-delà de ses espaces de conception et de développement, le système d'exploitation s'ancre dans des « lieux cruciaux » (Mounier-Kuhn, 2010) où il est implémenté, expérimenté, mis en œuvre et approprié à l'aune d'usages et de besoins locaux en dialogue avec la circulation transnationale des savoirs et savoir-faire du domaine. Comme illustré dans le premier volume de ce double numéro, le « Laboratoire d'informatique », centre de calcul du Cnam, rattaché en 1968 au département Mathématiques-Informatique créé dans l'établissement la même année, est un de ces lieux. Point de rencontre et d'expérimentation des membres fondateurs du projet de recherche qui mènera à la création du laboratoire Cédric, il est aussi un point de conflit entre différents services se disputant le recours aux machines pour des usages pédagogiques, scientifiques et administratifs. L'acquisition et l'utilisation des machines et de leurs systèmes, il est vrai, fournissent la trame d'une histoire non pas seulement des techniques mais aussi des usages. Fondé à l'origine sous le nom de « Laboratoire de calcul numérique », ce centre abrite d'abord des machines « sans systèmes », les machines mécanographiques à cartes perforées du constructeur français Bull, qui sont en priorité utilisées par les élèves-ingénieurs de la chaire de

Moteurs et font du calcul pour les différents laboratoires[3]. Le premier calculateur électronique est équipé en 1963, fonctionnant sur bandes perforées, à la faveur d'un élargissement des fonctions et des ambitions de ce « laboratoire de calcul » : il abrite alors des activités de traitement de la paie de l'établissement, et est mis à disposition d'enseignements scientifiques, alors qu'est créé le cours de « Machines Mathématiques », chaire sur laquelle sera élu Paul Namian en 1966. L'emblématique ordinateur IBM 360/30 est installé dès 1968, consacrant l'arrivée des machines à systèmes au Cnam – et l'on commence à parler de « salle ordinateur ». La machine continue à traiter les données à partir de cartes perforées mais elle est à présent pourvue de mémoire et de périphériques (non seulement les imprimantes, mais aussi des disques et bandes magnétiques). Le laboratoire d'informatique, son nom à partir de 1969, resserre alors ses fonctions autour d'un service à l'usage prioritaire des informaticiens, actant le rattachement au département Mathématiques-Informatique. Les langages de programmation deviennent un objet et un outil d'enseignement avec, dès le début de la décennie suivante, les travaux du groupe d'enseignement du CREEM (Centre de Recherche et d'Expérimentation pour l'Enseignement des Mathématiques) ; ce dernier s'installe sur le 360/30 avec lequel le dialogue s'établit via des consoles alpha-numériques (IBM-241) ainsi que sur une machine pédagogique dédiée à l'enseignement par interface interactive, la MITSI-2003 (Monitrice d'Instruction Technique et Scientifique). Une deuxième salle des ordinateurs est installée en 1973 qui systématise l'usage des mini-ordinateurs pour le traitement des travaux, notamment à travers des réseaux locaux (liaisons des machines avec des télétypes) et à distance (télétraitement avec le grand Centre de calcul d'Orsay, le CIRCÉ). Au même moment, un ordinateur Modular One va permettre de faire des expérimentations plus poussées de transferts de données en réseau entre des machines hétérogènes. La fin de la décennie voit arriver une nouvelle génération de mini-ordinateurs qui vont accélérer l'usage d'ordinateurs à systèmes innovants, flexibles, répondant aux usages spécialisés d'une diversité d'utilisateurs. En 1979, le laboratoire acquiert la machine PDP-11/70 du constructeur américain Digital Equipement (DEC), non sans difficultés à un moment où la politique protectionniste du gouvernement français en matière d'équipement informatique est encore en vigueur au crépuscule du Plan Calcul. Ce mini-ordinateur,

3 La liste de machines citées ici a été largement documentée, et commentée par Daniel Lippmann sur une page web aujourd'hui disparue [URL d'origine : http///ww2.cnam.fr/smdc/indexlabo.html]. Une copie du site a été conservée par les auteurs. Nous lui sommes redevable d'un travail d'archives pionnier sur l'informatique au Cnam et tenons à le remercier ici. Cette rapide histoire des machines et systèmes du Laboratoire d'informatique du Cnam est à mettre en perspective avec au moins deux textes publiés dans le premier volume de ce double numéro : l'entretien avec Gérard Florin, sous-directeur du Laboratoire d'informatique entre 1974-1975 et 1984-1985 ; et l'article de Paloque-Berges et Petitgirard, « L'Équipe Systèmes (1975-1980) et la genèse de la recherche en informatique au Cnam ».

mythique dans l'histoire des systèmes et des réseaux est officiellement dédié à une cellule informatique de gestion, pour le traitement de la paie et autres tâches de l'administration centrale, mais est utilisé en perruque la nuit par les informaticiens du Cnam pour leurs propres expérimentations. Alors que les « vieilles » machines, comme le Modular One et le IBM 360/30 sont abandonnées, des machines innovantes et à la réputation internationale comme le PDP font leur apparition, en particulier un autre ordinateur de DEC, le VAX-11/780. Équipés de terminaux vidéo, elles permettent d'achever et de pérenniser les premières dynamiques d'utilisation interactive des machines par les utilisateurs spécialisés. Le PDP et le VAX sont deux machines fondamentales dans la montée en recherche des informaticiens du Cnam, car elles sont précisément celles sur lesquelles les problématiques des systèmes pourront être enseignées, traitées et expérimentées.

Les machines, dans cette histoire, sont comme nous l'avons dit une des trames de lecture possible d'une histoire de l'informatique qui s'intéresse aux systèmes et applications logicielles tout autant qu'aux acteurs humains et institutionnels. En effet, privilégier tel ou tel équipement se fait dans le cadre de contraintes fortes sur le plan de politiques européennes économiques en matière de sciences et technologies, en dialogue et/ou en opposition avec la domination industrielle américaine symbolisée par IBM (voir entre autres Paju et Haigh, 2015 ; Cortada, 2002 ; Campbell-Kelly et Aspray, 1996). Mais la faveur donnée à certaines machines est aussi le fait de choix spécifiques d'utilisateurs liés aux engagements et croyances des informaticiens dans des cultures techniques et scientifiques données, définies par des équipements et des styles de programmation. La question des croyances idéologiques autour de l'usage scientifique des machines est également en dialogue avec des problématiques épistémologiques : il n'est pas anodin que l'expression de « machines mathématiques » soit définitivement abandonnée en France dans le cours des années 1970, pour le terme de « systèmes informatiques ».

Ces interrogations nous ont amenés à nous pencher sur un système symbolique de ces transformations techniques, scientifiques, politiques et culturelles de l'informatique dans les années 1970, le système Unix, auquel une grande partie de ce numéro est consacrée[4]. Unix est his-

4 On constatera que la graphie du terme varie au gré des articles : nous respectons le choix des auteurs en la matière, car il fait signe également vers l'histoire du nom du système. En minuscules, il désigne le système tel que conçu à l'origine par Ken Thompson et Dennis Ritchie, et nommé par Brian Kernighan (d'abord sous la forme d'Unics) en référence au système sur lequel ils travaillaient auparavant, Multics (genèse discutée dans plusieurs des articles du dossier, dont ceux de Warren Toomey et Thomas Haigh). Déposé comme marque par AT&T (voir notamment l'article de Clement Cole qui évoque la question des contraintes juridiques et commerciales imposées au système par AT&T), il prend alors des majuscules, et il est souvent cité dans la littérature des années 1980 avec sa mention commerciale, UNIX™, à la fois pour se protéger de poursuites de l'entreprise et pour s'en moquer, alors que le système a été largement adopté par les milieux de la programmation système, et a durablement influencé les communautés de logiciels libre et open source.

toriquement associé aux développements et à la promotion de la recherche sur les systèmes ouverts et les réseaux informatiques. Il se déploie dans cette décennie et la suivante à la croisée des mondes industriel et académique.

Unix est né en 1969 au sein des Bell Labs, laboratoire de R&D du géant des télécoms américain AT&T. Nous avons traduit le texte de W. Toomey consacré à la genèse du système Unix (« de A à Z ») dans ce contexte, et les efforts récents en archéologie logicielle pour retrouver les premières versions d'Unix sur la célèbre machine DEC, le PDP-7. Focalisé sur l'implémentation pas à pas du système, le texte circonstancie au mieux les recherches de Ken Thompson et Dennis Ritchie, ses concepteurs. Au-delà de sa conception, c'est la circulation et l'évolution du système qui font date. Le système ne peut pas être commercialisé par l'entreprise suite à un décret antitrust datant de 1956. Il est donc diffusé dans les universités contre un faible coût de distribution et de licence. Une équipe d'informaticiens de Berkeley spécialisés dans le logiciel et les systèmes s'en empare (Berkeley Software Development – ou BSD), initiant une suite de versions dont l'appropriation dans le milieu académique international fera date et constituera une étape fondamentale de la construction d'une culture des logiciels « ouverts », partageant et modifiant collectivement le code source des programmes informatiques (Kelty, 2008). Le succès d'Unix, d'abord implémenté sur les mini-ordinateurs, puis sur les stations Sun de micro-informatique avant de devenir plus largement compatible avec le parc informatique, a été encouragé par ses qualités de système ouvert, multi-utilisateurs et multitâche favorisant l'expérimentation logicielle. Dans ce dossier, C. Cole, un informaticien qui a participé aux développements d'Unix aux États-Unis depuis les années 1980, souligne à quel point ces caractéristiques ont été décisives pour que s'organise une communauté Unix, qui a permis la prolifération du système face aux efforts des industriels de l'informatique d'imposer leurs systèmes propriétaires.

Dans le contexte européen, en particulier en France, cette innovation américaine a été non seulement adoptée, mais aussi transposée et adaptée. Notre dossier donne une place aux développements orchestrés au sein du groupe Bull, dans les années 1980 et au début de la décennie suivante. M. Élie et P. Picard, deux ingénieurs ayant fait une partie de leur carrière chez Bull et dans le domaine des systèmes ouverts, montrent dans quelles circonstances Bull est sorti de son schéma de systèmes propriétaires, pour élaborer une gamme de machines Unix. De son côté J. Talbot retrace cette histoire du point de vue du groupe d'informaticiens de Bull en charge des développements Unix à Grenoble entre 1982 et 1992 : ingénieur informaticien, il a été membre actif de ce groupe et montre dans son texte l'articulation entre ce contexte local, la stratégie nationale d'ensemble du groupe Bull et les efforts de développement d'Unix à l'échelle internationale.

Dans les années 1970, alors que l'informatique se diversifie et devient plus accessible, émergent des communautés d'intérêts et de pratiques nationales et internationales relativement indépendantes des constructeurs industriels ou des établissements où les acteurs travaillent. Parmi eux, les groupes d'utilisateurs d'Unix, réseau international. Dans le même article de C. Cole on en trouvera d'ailleurs une illustration à travers la naissance et l'activité de l'association Usenix. En proposant une contextualisation de l'apparition d'Unix selon une perspective nouvelle, T. Haigh en revient au rôle des pratiques et communautés du développement du logiciel en amont des réseaux associatifs autour d'Unix : à la fois en montrant que le courant de l'*open source* a en réalité une longue histoire dans le monde industriel (remontant aux efforts de structuration de la communauté SHARE au sein d'IBM notamment) ; et en pointant que le développement para-académique (ou quasi-académique) d'Unix est assez représentatif des pratiques de développement logiciel des années 1970, qui ont tenté, toujours selon la même ligne historique, de mettre en commun des routines et des programmes si laborieux à élaborer.

L'histoire d'Unix s'interroge ainsi à plusieurs échelles : (1) l'innovation, les milieux et les pratiques innovantes des laboratoires de recherche et de développement dans les universités et des entreprises ayant participé au développement ou à la diffusion d'Unix ; (2) celle des orientations scientifiques et politiques concernant les systèmes ouverts (normes et standards) ; (3) au niveau des relations internationales voire géopolitiques et géostratégiques (relations entre utilisateurs d'Unix et constructeurs d'ordinateurs en Europe et à l'international, rôle des embargos américains dans la diffusion du système et des machines qui le supportent...). Ce jeu d'échelles est à replacer au sein de la diffusion phénoménale du système à travers les milieux scientifiques et de la R&D. Les groupes et clubs d'utilisateurs ont joué un rôle crucial dans l'émergence du développement collaboratif autour de logiciels (Mounier-Kuhn, 2010 ; Haigh, 2002), en tant que regroupements d'utilisateurs professionnels spécialisés dans un type de matériel ou de logiciel initiés par les constructeurs qui souhaitent former les ingénieurs et techniciens à l'utilisation d'une machine spécifique, et s'étant peu à peu autonomisés de ces constructeurs. En ceci, le rôle des utilisateurs spécialistes, les développeurs et ingénieurs systèmes en particulier, a été déterminant. Ils sont non seulement pionniers en matière de conception, expérimentation, et développement, mais aussi ouvriers de la structuration d'un milieu de collaboration par des tâches moins nobles mais tout aussi importantes dans le processus d'innovation : l'implémentation, l'adaptation, la mise à jour, la maintenance, la sécurisation, la réparation des matériels et logiciels formant un parc d'équipement informatique international.

C'est sur quelques réflexions sur l'émergence et la consolidation d'un

esprit Unix, compris comme l'une des expressions historiques d'une culture technique construite autour des systèmes informatiques, que nous conclurons. Les réseaux d'informaticiens utilisateurs d'Unix, d'ailleurs présents au Cnam au sein du laboratoire d'informatique, fédèrent des solidarités socio-professionnelles qui ne sont pas tout à fait alignées avec les politiques de développement scientifique et technique traditionnelles des pays industrialisés – voire qui revendiquent leur indépendance face à ces politiques, leur préférant des réseaux de solidarité technique transnationaux. Or, ces réseaux se construisent et se développent notamment à partir de références croisées qui peuvent être d'ordre idéologique, culturel, aussi bien que scientifique et technique. Se façonne alors un ethos de la communauté réunie autour d'Unix que les acteurs mêmes nomment « philosophie Unix », dont les référents culturels relèvent d'un style de programmation (formulé en termes techniques et esthétiques) aussi bien que d'une éthique revendiquée dans le rapport à l'informatique (autonomie, ouverture…) matérialisée dans les pratiques et choix techniques des utilisateurs. Il faut préciser aussi que la diffusion d'Unix est souvent décrite dans une logique d'adoption, parfois d'appropriation, mais plus rarement à l'aune des résistances que le système a pu rencontrer pour des raisons de limitations techniques ou de contraintes économiques et politiques en matière d'équipement technologique ; les textes sur la place d'Unix au sein de Bull dans ce volume laissent entrevoir certaines de ces contraintes. Enfin, comme le montre le texte de L. Bloch dans ce numéro, il faut insister sur le poids des croyances, en interaction avec les choix scientifiques et préférences technologiques, dans la constitution de la communauté unixienne. Observateur et analyste des évolutions de l'informatique, L. Bloch a été ingénieur système et réseau, notamment au Cnam, à l'Insee et à l'Institut Pasteur, et a participé aux premières années d'Internet en France au tournant des années 1990. Il propose une lecture des logiques d'appartenance, de fidélité, pointant une forme de prophétisme informatique dans le développement d'Unix. On retrouve ici une analogie religieuse, qui fait écho à bien d'autres phénomènes quasi-cultuels traversant l'histoire des sciences et techniques.

La popularité d'Unix dans l'histoire de l'informatique suppose l'appropriation et la construction d'une culture Unix non seulement consolidée précocement mais perdurant aussi jusqu'à aujourd'hui. La mémoire des utilisateurs Unix est vive, portée très tôt par des enquêtes journalistiques dans les milieux de la recherche et du développement (Salus, 1994), ainsi qu'une pratique de sauvegarde patrimoniale avancée (Toomey, 2010). L'héritage d'Unix est constitué non seulement de la longue série de versions et d'applications dérivées du système, jusqu'aux systèmes mobiles Android aujourd'hui ; mais aussi de la mémoire de ses développements, toujours discutée aujourd'hui dans les forums de spécialistes, et objets de travaux de reconstitu-

tion[5]. Au final, bien qu'encore peu en tant que tel couvert par les études de sciences humaines et sociales, le système Unix est un référent important dans l'histoire de l'informatique, en particulier dans le champ des logiciels libres (Kelty, 2008) et des réseaux (Paloque-Berges, 2017), ainsi que dans la philosophie des langages et des temps de la programmation (Mélès, 2013, 2017).

5 L'association The Unix Heritage Society, dirigée par Warren Toomey, auteur dans ce dossier, abrite non seulement un forum de liste très actif, mais aussi des travaux d'archéologie des systèmes et logiciels Unix. [URL: http://www.tuhs.org/].

Bibliographie

Campbell-Kelly M. (2003). *From airline reservations to Sonic the Hedgehog: a history of the software industry.* Cambridge, Mass. : MIT Press.

Campbell-Kelly M., Aspray W. (1996). *Computer: A History of the Information Machine*. New York : Basic Books.

Cortada, J. W. (2002). « IBM Branch Offices: What They Were, How They Worked, 1920s-1980s ». *IEEE Annals of the History of Computing* 39.3, pp. 9-23.

Hashagen U., Keil-Slawik R. & Norberg A. L. (eds.) (2002). *History of computing: software issues. International conference of the history of computing* (Actes de colloque de ICHC 2000, April 5-7, 2000, Heinz Nixdorf MuseumsForum, Paderborn, Germany). Berlin : Springer-Verlag.

Haigh T. (2002). « Software in the 1960s as Concept, Service, and Product ». *IEEE Annals of the History of Computing* 24.1, 2002, pp. 5-13.

Kelty C. M. (2008). *Two Bits: The Cultural Significance of Free Software.* Durham : Duke University Press Books.

Mélès B. (2013). « Unix selon l'ordre des raisons : la philosophie de la pratique informatique ». *Philosophia Scientiæ* 17 (3), pp. 181-198.

Mélès B. (2017). « Temps et activités selon Unix ». *Réseaux* 206/6, pp. 125-153.

Mounier-Kuhn P.-É. (2010). « Les clubs d'utilisateurs : entre syndicats de clients, outils marketing et 'logiciel libre' avant la lettre ». *Entreprises et histoire 3,* 2010, pp. 158-169.

Neumann C., Paloque-Berges C. & Petitgirard L. (2016). « 'J'ai eu une carrière à l'envers' : entretien avec Claude Kaiser, titulaire

de la chaire d'Informatique-Programmation du Conservatoire National des Arts et Métiers ». *Techniques et Sciences Informatiques (TSI)* 35 (4-5), pp. 557-570.

Paju P. & Haigh T. (2015). « IBM Rebuilds Europe: The Curious Case of the Transnational Typewriter ». *Enterprise & Society,* Vol. 17, Number 2, pp. 265-300.

Paloque-Berges C. (2017). « Mapping a French Internet experience: a decade of Unix networks cooperation (1983-1993) ». *In* G. Goggin et M. McLelland (eds.), *Routledge Companion to Global Internet Histories.* New-York : Routledge, 153-170.

Salus P. H. (1994). *A Quarter Century of UNIX.* Addison-Wesley, Reading.

Toomey W. (2010). « First Edition Unix: Its Creation and Restoration ». *IEEE Annals of the History of Computing* 32 (3), pp. 74-82.

Qu'est-ce qu'un système d'exploitation ? Une approche historique

Maarten Bullynck
Centre de recherches historiques : Histoire des pouvoirs, savoirs et sociétés (EA 1571), Université Paris 8.

Résumé

Les débuts des systèmes d'exploitation sont souvent obscurcis par leur proximité aux systèmes de programmation et ont été minimisés dans la suite des discussions des années 1960 entre les défenseurs du traitement par lots et les partisans du partage de temps. Une étude plus détaillée et plus systématique des systèmes de contrôle entre 1954-1964 montre pourtant qu'il existe une variété d'idées, de philosophies et d'implémentations qui ont souvent préparé la voie à des systèmes ultérieurs. Cet article offre d'abord une synthèse de développements pertinents en matériel et en logiciel et propose ensuite une taxonomie de systèmes de contrôle pendant la période 1954-1964. Enfin, les origines du terme operating system et sa dissémination sont retracées dans la communauté des utilisateurs des machines d'IBM.

Mots-clés : systèmes d'exploitation, histoire de l'informatique, organisation du travail informatique, partage de temps, programmation automatique.

Un système d'exploitation est aujourd'hui devenu indispensable : on ne peut guère s'imaginer un ordinateur sans un système d'exploitation qui gère et façonne l'accès à l'ordinateur et à ses périphériques, et sans que l'interaction entre l'ordinateur et ses utilisateurs ne passe par cette interface parfois devenue invisible[1]. Pourtant, quand les premiers ordinateurs sont apparus après la seconde Guerre Mondiale, il n'y avait rien de tel. Il a fallu une décennie avant que les premiers essais de système d'exploitation ne soient formulés et implémentés. Il a fallu une décennie supplémentaire pour que l'idée soit généralement acceptée et que presque tous les ordinateurs soient vendus ou loués en incluant un système d'exploitation. Il faut attendre la fin des

1 Cet article est une adaptation libre d'un article qui apparaîtra dans un volume spécial de HAPOP-3.

années 1960, avec le développement des systèmes d'exploitation OS/360 par IBM et de Multics par une équipe associant le M.I.T., les Bell Labs et General Electric, pour que des principes généraux et des cadres théoriques pour les systèmes d'exploitation soient mis en place. L'émergence des systèmes de partage de temps (*time-sharing*) est considérée traditionnellement comme un tournant dans l'histoire des systèmes d'exploitation. Les discussions parfois acérées entre les partisans du *time-sharing* et les défenseurs du traitement par lots (*batch processing*) dans les années 1965-1975[2] sont devenues légendaires et font partie intégrante de l'histoire de l'informatique. Mais ces discussions, aussi importantes soient-elles pour repenser la relation entre ordinateur et utilisateur, et révélatrices aussi d'un « *combat gargantuesque pour dominer le marché du logiciel qui se chiffre en milliards* »[3], ont biaisé l'historiographie des systèmes d'exploitation.

L'histoire classique de ces systèmes part de l'ordinateur sans système d'exploitation, passe par le traitement par lots pour arriver enfin à la multiprogrammation ou le *time-sharing* moderne[4]. C'est une histoire où une configuration variable de programmeurs, opérateurs et ingénieurs travaillant avec un ordinateur est rationalisée à l'aide du traitement par lots. Une nouvelle configuration, qui automatise partiellement le programmeur et l'opérateur, se met en place : le travail des programmeurs est séparé du travail opératoire. Les programmeurs, travaillent dans leurs bureaux, écrivent leurs programmes puis les apportent à l'ordinateur et ses opérateurs. Les programmes sont mis en lots et attendent leur exécution, qui est gérée par l'opérateur et aidée par un système d'exploitation. Cette histoire tourne aussi, sans l'évoquer explicitement, autour de l'entreprise IBM. Le traitement par lots a débuté sur des machines d'IBM et quand commencent les débats entre partisans du *batch-processing* et du *time-sharing* au début des années 1960, IBM sera lent à repérer le *time-sharing* et on lui reprochera de rester plutôt fidèle au traitement par lots. Ce sont tous des fils pertinents de l'histoire des systèmes d'exploitation, mais ils cachent la variété et la complexité des systèmes issus de besoins et visions très différents des utilisateurs de la période allant de 1954 à 1964. Cette période n'est pas seulement l'ère des systèmes de traitement par lots. Au contraire, il existait plusieurs modes possibles d'utilisation d'un ordinateur et plusieurs concepts pour automatiser en partie cette utilisation. En outre, l'idée même d'un système d'exploitation n'était pas stabilisée.

Dans ce qui suit on explore des questions et problèmes qui ont stimulé

2 Un échantillon caractéristique de cette discussion se trouve dans le numéro spécial de Septembre 1965 du journal *Computers and Automation*.

3 « […] *a gargantuan contest to dominate the multibillion dollar software industry* » (Sackman, 1970, p. 8).

4 Ce récit classique se trouve chez (Ceruzzi, 2003, pp. 96-101), (Tanenbaum, 2001, pp. 6-18) ou (Krakowiak, 2013, 2014). Le livre de Brinch-Hansen (2001) suit aussi cette chronologie, mais sa présentation met en évidence que les systèmes et leurs « philosophies » se développent en parallèle.

les débuts des systèmes d'exploitation[5]. Cette démarche permet d'explorer des couches historiographiques et épistémologiques qui approfondissent le récit classique tout en mettant en valeur des développements en parallèle et en restant focalisé sur cette question : qu'est-ce qu'un système d'exploitation ?

Prérequis technologiques pour les systèmes d'exploitation (1954-1964)

Débuts dans les années 1950

L'idée même que la machine est en charge de sa propre programmation est en fait implicite dans le modèle de l'ordinateur à programme enregistré (*stored-program computer*). Parce que l'ordinateur calcule beaucoup plus vite que l'homme, l'ordinateur doit contrôler et gérer l'exécution des programmes. L'étape suivante conduirait logiquement à l'idée qu'un programme peut contrôler et gérer les autres programmes. Pourtant, dans les années 1946-1956 peu d'ordinateurs étaient capables d'implémenter cette idée : tout au plus, des routines préparatoires ou des routines d'amorçage (*bootstrapping routines*) étaient disponibles. Au milieu des années 1950, cette situation change pour de nombreuses raisons. La diversification de l'utilisation de l'ordinateur, en particulier son usage croissant pour le traitement des données (*data processing*), combiné avec des développements technologiques et des développements logiciels font naître une variété de systèmes pour exploiter un ordinateur.

Côté technologie, les tambours magnétiques (*magnetic drum memory*) deviennent populaires en tant que mémoire de travail, et parfois comme mémoire de stockage. Les tores magnétiques (*magnetic ferrite core memories*) sont développés au M.I.T. au début des années 1950, technologie qui rend possible une mémoire de travail plus stable, plus rapidement accessible. Les tambours magnétiques sont surtout populaires dans les années 1950, les tores magnétiques par contre, étant plus chers à leur début, ne se répandront vraiment que dans les années 1960. Les bandes magnétiques constituent la dernière technologie importante. Cette technologie, proposée dès 1951, permet une mémoire de stockage plus rapide et plus importante. Lentement, les bandes magnétiques vont remplacer les cartes perforées et les bandes de papier perforées, même si jusque dans les années 1960 ces médias de stockage sont encore utilisés en parallèle des bandes magnétiques. Les tambours magnétiques et les bandes magnétiques permettent un accès plus rapide aux données enregistrées et facilitent donc l'accès à un grand nombre de routines (une bibliothèque de routines). Comparé aux cartes perforées, cet accès n'est pas seulement plus rapide mais aussi plus automatisable et donc moins assujetti à

5 Cette analyse s'appuie sur une étude systématique des premiers systèmes qui sera publiée ailleurs.

l'intervention humaine. Évidemment, le mode d'accès aux données est soumis aux propriétés physiques des supports : la lecture et l'écriture se font de manière séquentielle (pour les bandes) ou cyclique (pour les tambours). Un accès direct (*random access*) deviendra possible seulement dans les années 1960, avec les tores magnétiques et les disques durs (*disk drive*).

L'introduction des mémoires tampons (*buffer memory*) pour la communication entre le processeur central et les organes d'entrée et de sortie constitue une autre avancée technologique. Avant ces mémoires, le processeur devait attendre la fin d'une lecture ou d'une écriture, ou d'une communication avec un dispositif d'entrée ou de sortie, pour passer à l'étape suivante. Cette forme de communication limitait la vitesse du processeur et pouvait même la réduire à la vitesse de fonctionnement des organes d'entrée ou de sortie. Si on sait par exemple que sur l'ENIAC le processeur pouvait opérer 5 000 additions par seconde, le lecteur de cartes perforées ne pouvait lire que 2 cartes par seconde : il est facile d'imaginer que cette inadéquation entre vitesses pouvait ralentir sérieusement l'opération d'un ordinateur. De nombreuses stratégies avaient été élaborées pour éviter ce blocage, utilisant des bandes magnétiques comme mémoire tampon, ou même utilisant tout un ordinateur pour traiter les communications avec des périphériques entrée et sortie. Mais avec l'introduction systématique des mémoires tampon pour entrée et sortie, « *l'entrée et la sortie ont quitté le domaine des vitesses mécaniques pour accéder au domaine des vitesses électroniques* »[6].

L'expansion du stockage rapide pour les programmes va de pair avec le développement du logiciel (*software*). Les routines programmées sont cumulativement rassemblées dans une bibliothèque, stockées dans une mémoire externe. Entre utilisateurs d'un même type d'ordinateur s'installent des procédures et des usages de partage et de recyclage. La fondation des groupes d'utilisateurs comme SHARE pour les utilisateurs du IBM 701 ou de USE pour les utilisateurs du UNIVAC 1103 (tous les deux en 1955) marque le moment où le logiciel devient un article de valeur. Des groupes d'utilisateurs se réunissent régulièrement pour partager des programmes et pour discuter des pratiques de programmation. Si les programmes dans ces groupes étaient partagés gratuitement, la deuxième moitié des années 1950 voient également les débuts d'une industrie du logiciel (Cerruzzi, 2003, pp. 79-108). Des sociétés comme System Development Corporation (SDC, 1957) ou Computer Sciences Corporation (CSC, 1959) louaient leurs services aux grands projets du gouvernement ou aux grandes entreprises (Campbell-Kelly, 2001, pp. 29-56).

6 « *With the advent of this phase [I/O buffer memory], input-output was taken out of the domain of mechanical speeds and placed in the domain of electronic speeds* » (Bauer, 1958).

Dans ce contexte, les premiers langages de programmation sont développés, des bibliothèques de routines s'accumulent, et on voit aussi l'apparition de systèmes de programmation qu'on peut appeler, *a posteriori*, des systèmes d'exploitation. L'un des premiers systèmes, qui sera très influent, a été le *Comprehensive System of Service Routines* (CSSR) développé au Lincoln Lab du M.I.T. pour leur ordinateur Whirlwind (1953). Ce système sera plus tard le fond sur lequel SDC va ériger son système de programmation pour le projet SAGE. Un autre type de systèmes importants, les moniteurs de séquences de programmes (*monitor program*) faits pour enchaîner automatiquement l'exécution de programmes, sont développés au cœur de la communauté de SHARE. Ces moniteurs deviendront plus tard les noyaux pour les systèmes de traitement par lots (*batch processing*), très typiques pour les installations commerciales et scientifiques des machines IBM dans les années 1960.

Deux innovations ponctuelles, l'une en technologie, l'autre en logiciel, sont cruciales pour l'évolution des systèmes d'exploitation. D'abord, en 1956, l'interruption (*interrupt*) était introduite sur le ERA 1103A sur la demande d'un utilisateur, la NSA[7]. L'interruption pouvait interrompre la marche de la machine pour faire communiquer le processeur central avec un périphérique. En utilisant l'interruption, les interruptions manuelles effectuées par l'opérateur humain pouvaient être (partiellement) automatisées. Par cette voie, des moniteurs plus compliqués étaient envisageables et la multiprogrammation, et plus tard le partage de temps, devenait possible. Puis, en 1957 est sorti FORTRAN (pour FORmula TRANslation), aboutissement du travail d'un groupe de programmeurs pour IBM visant à créer un langage scientifique de programmation. En tant que premier langage de programmation complet[8], FORTRAN (puis FORTRAN II) devint vite populaire et presque indispensable pour beaucoup d'installations d'ordinateur. Pour beaucoup de systèmes de programmation existants, l'apparition de FORTRAN a mené à des réécritures considérables afin de pouvoir l'intégrer. Un bon nombre de systèmes d'exploitation autour de 1960 sont (re)développés par de grands utilisateurs institutionnels pour inclure FORTRAN, par exemple le FORTRAN Monitor System (FMS, 1959) développé par North American Aviation, ou BESYS-3 (1960) par Bell Labs, UMES (1959) par l'université de Michigan ou encore le RAND-SHARE Operating System (1962) développé par la RAND Corporation.

7 Cet ordinateur est parfois aussi connu sous le nom de Scientific Univac 1103, et il fut d'abord développé pour la NSA sous le nom Atlas II.

8 Il y a d'autres langages de programmation développés avant ou en parallèle de FORTRAN, voir (Knuth & Pardo, 1977), mais ou bien leur utilisation fut limitée, ou bien il leur manquait souvent des fonctionnalités importantes, comme par exemple des commandes pour programmer les périphériques.

Changements au milieu des années 1960

Les années entre 1962 et 1964 marquent la fin d'une première phase dans le développement des systèmes d'exploitation. Les « grands » projets de systèmes d'exploitation comme OS/360 d'IBM ou Multics, et en particulier l'émergence des systèmes à temps partagé, symbolisent ce tournant, même s'ils sont plutôt les traits les plus visibles d'une évolution plus générale. Cette évolution est portée, d'une part, par le développement graduel de la multiprogrammation, et d'autre part, par l'introduction de nouveaux types de mémoire de stockage. La multiprogrammation, en essence, repose sur l'idée que plusieurs programmes peuvent être exécutés en même temps sur une machine, ce qui rompt avec le traitement séquentiel inhérent à l'architecture de von Neumann, où une instruction est exécutée seulement après que la précédente soit terminée. En pratique, la multiprogrammation n'est souvent qu'une répartition du temps du processeur central sur plusieurs programmes. Le processeur de l'ordinateur traite encore séquentiellement les programmes, mais, en utilisant des mémoires tampons (*buffer memories*), des programmes attendent leur tour, ou sont interrompus pour être repris plus tard. L'ordonnancement (*scheduling*) organise l'ordre d'exécution de (morceaux) de programmes. L'interrupteur (*hardware interrupt*) a rendu possible les premières tentatives de multiprogrammation, et l'introduction des mémoires tampon pour entrée et sortie a fait proliférer la multiprogrammation sous des formes variées, dont l'une des plus extrêmes est le partage de temps (*time-sharing*). Pourtant, la transition d'une mémoire de stockage séquentielle à une mémoire vive à accès direct (*random access*) a peut-être eu l'influence la plus profonde sur l'implémentation des systèmes d'exploitation dans les années 1960. Cette transition a permis d'abandonner la logique séquentielle des bandes magnétiques et d'initier l'accès direct. Le disque dur du RAMAC d'IBM (1956) est le premier exemple d'une mémoire vive. Les disques IBM 1405 et IBM 1301 (1961-1962), développés pour être utilisés sur l'IBM 1410 et la ligne IBM 7000, ont été les systèmes les plus répandus.

Mais il n'y a pas seulement des avancées technologiques. Entre 1962 et 1964 il semble que presque tous les producteurs d'ordinateurs ont repris l'idée d'un système d'exploitation et l'ont développé pour l'inclure dans le paquet ordinateur-logiciel qui était loué ou vendu[9]. Avant 1960, le développement des systèmes d'exploitation était assuré en majorité par les utilisateurs des systèmes d'ordinateurs et dans quelques contextes bien particuliers (surtout des projets de recherche financés par le militaire). Après 1960, ce sont les entreprises qui prennent en main le logiciel (*software*), elles embauchent des programmeurs et investissent dans

9 C'est également au début des années 1960 que les premiers articles de synthèse sur les systèmes d'exploitation apparaissent, notamment (Orchard-Hays, 1961, pp.290-294 ; Mealy, 1962).

la conception d'outils de programmation pour leurs machines. Ces outils incluent des bibliothèques de routines, des (macro) assembleurs, des compilateurs pour des langages de programmation (comme FORTRAN ou COBOL), des langages de programmation, des outils de *debugging*, mais aussi des systèmes pour organiser des fichiers ou des données, des « routines de maître » (*master routines*) et des systèmes d'exploitation. Si on regarde quelques-unes des principales sociétés de systèmes informatiques, elles sortent toutes leur propre système d'exploitation entre 1962 et 1965 (voir Table 1). Certains de ces systèmes sont plutôt rudimentaires (comme BRIDGE de General Electric), d'autres sont des systèmes de traitement par lots classiques (comme BKS de Philco ou Scope de CDC), mais la majorité ont des options avancées de multiprogrammation combinées avec le traitement par lots. Le partage de temps, par contre, est encore en développement et existe avant 1966 surtout dans des contextes de recherche, par exemple CTSS sur un IBM 709 et plus tard sur un PDP-1 (M.I.T.), TS sur un PDP-1 (BBN), JOSS sur le Johnniac (RAND), le Dartmouth Time-Sharing System (Dartmouth College), le Cambridge multiple-access System sur le Ferranti Atlas (Cambridge), etc. Dans des installations commerciales, le partage de temps fait son apparition vers la fin des années 1960 quand IBM, GE, DEC, PDP, SDS et d'autres sociétés vont l'incorporer dans leurs systèmes d'exploitation.

Société	Ordinateur	Année	Système d'exploitation
Honeywell	H800	1961	Executive Monitor
Univac	Univac 1107	1962	EXEC I
Burroughs	D825	1962	AOSP
Burroughs	B5000	1962	Master Control Program
Philco	Philco-2000	1962	SYS; BKS
GE	GE-215/225/235	1962	BRIDGE
IBM	IBM 7090/7094	1962	IBSYS
CDC	CDC 1604	1962	CO-OP monitor system
CDC	CDC 3600	1963	SCOPE monitor system
Honeywell	Honeywell 1800	1963	ADMIRAL master monitor
GE	GE 625-635	1964	Comprehensive Operating Supervisor
RCA	RCA 3301	1964	Realcom System
DEC	PDP-6	1964	Supervisory Control Program
SDS	SDS 9000	1964	MONARCH

L'évolution d'un développement par l'utilisateur vers des systèmes faits maison par les sociétés elles-mêmes est la plus apparente chez IBM. Les premiers systèmes d'exploitation pour des machines IBM sont développés par des utilisateurs comme General Motors, North American Aviation, Bell Labs, Michigan University, etc. Ils s'appuyaient sur le partage d'idées et de programmes au sein de la communauté SHARE d'utilisateurs de machines IBM, mais ne recevaient pas de support direct d'IBM même. Lentement, IBM en tant que société s'est impliquée également. Le constructeur a aidé le développement de Share Operating System (1959) qui est né au cœur de SHARE. Plus tard, il va incorporer le FORTAN Monitor System, initialement développé par North American Aviation, dans son système de programmation FORTRAN pour le IBM 709/7090 (1960). Finalement, IBM commence à concevoir ses propres systèmes d'exploitation, d'abord IBSYS (à partir de 1962), puis OS/360 (à partir de 1965). En parallèle, le développement par les utilisateurs disparaît lentement, même si certains utilisateurs vont encore bricoler avec les systèmes des fabricants et les adapter à leurs besoins[10].

Qu'est-ce qu'un système d'exploitation ? Des philosophies différentes

Si on utilise le terme usuel en anglais pour système d'exploitation, *operating system*[11], on souscrit implicitement à l'idée qu'un système d'exploitation automatise la mise en opération de l'ordinateur et remplace, en partie, le travail d'un opérateur ou d'une opératrice humaine. On peut l'appeler la philosophie opérationnelle du système d'exploitation. En particulier, les opérations à exécuter manuellement sur un panneau de contrôle (*control panel*), ou panneau moniteur (*monitor panel*) ou panneau de supervision (*supervisory panel*) sont automatisées par l'*operating system*. Sur ces panneaux, l'opérateur pouvait agir quand il y avait des arrêts (après l'exécution d'un programme ou après la fin d'une lecture ou d'une écriture par un périphérique, ou quand il faut attendre que l'opérateur prenne la bande magnétique de la bibliothèque de routines dont on a besoin) ou des interruptions (quand un programme ou un périphérique ne fonctionne pas comme prévu, ou quand l'instruction ne peut pas être exécutée), ou d'autres signaux encore. L'*operating system* automatise les réponses de l'opérateur à ces arrêts, interruptions et signaux. Par cette auto-

10 Un exemple est Thompson-Ramo-Woolridge qui va développer un système d'exploitation à partir de IBSYS en 1962 (Nelson, 1969).

11 La plupart des langues modernes utilisent une variante de *operating system*, mais pas tous, par exemple le français (système d'exploitation), l'allemand (*Betriebssystem*), le néerlandais (*besturingssysteem*) ont d'autres termes.

matisation, il devient possible que toute une séquence, un lot (*batch*) de programmes, puisse être accomplie sans interruption entre deux arrêts, et non plus seulement un programme à la fois : d'où le nom « traitement par lots » (*batch processing*) pour cette première génération de systèmes d'exploitation. Ce type de système pouvait réduire le temps inutilisé de l'ordinateur et accélérer le chargement des programmes. En outre, certaines erreurs humaines pouvaient être évitées et on pouvait automatiser les processus de chargement et de traduction, une fois qu'ils étaient standardisés et codés dans un format fixe. Dans ce contexte, on dit souvent à l'époque que le système d'exploitation fait le « ménage » (*housekeeping*).

L'automatisation partielle de l'opérateur humain a mené aussi à une autre configuration et opération dans les centres de calcul. Traditionnellement, on parle de la transition d'*open shop* (opération ouverte du centre de calcul) à *closed shop* (opération fermée)[12]. Dans la configuration *open shop*, on emmenait son programme au centre de calcul, laissait l'opérateur mettre le programme sur l'ordinateur, et après exécution du programme, on pouvait prendre les résultats et retourner dans son bureau. La configuration *closed shop* par contre réduisait le contact avec l'ordinateur et son opération, l'opérateur humain recueillait les programmes et les mettait en lots. On devait attendre l'exécution du lot dans lequel était son programme pour pouvoir chercher les résultats dans le centre de calcul. La configuration *closed shop* est donc intimement liée à la philosophie de traitement par lots[13].

On cite souvent 1956 comme l'année de naissance pour les systèmes d'exploitation du type traitement par lots, mais la généalogie commence un peu plus tôt avec des systèmes comme le 701 Monitor (1955) de Owen Mock à North American Aviation (Mock, 1987) ou le Supervisor pour l'IBM 702 de Bruse Moncreiff à RAND (Moncreiff, 1956). L'idée mûrit avec le développement en 1956 du General Motors-North American Aviation Monitor (sous l'acronyme GM/NAA Monitor) pour un IBM 704, moniteur qui sera partagé dans la communauté SHARE (Patrick, 1987). Le noyau de ce système, le Mock-Donald monitor, sera recyclé, adapté, actualisé et implanté dans des systèmes d'exploitation plus complexes et plus ambitieux comme le SHARE Operating System

12 Les termes *open shop* et *closed shop* sont empruntés aux pratiques des syndicats aux États-Unis. En *closed shop*, les ouvriers devraient être syndiqués pour faire une tâche et il y avait des restrictions sur le type de tâche qu'un ouvrier pouvait faire.

13 Parfois, une autre interprétation est donnée à *open shop* versus *closed shop*. Dans cette interprétation, le *closed shop* est la situation où seulement les opérateurs et les programmeurs en code machine peuvent utiliser la machine, parce que les autres utilisateurs ne savent pas écrire des programmes en code machine ou assembleur. L'*open shop* est alors le cas où les utilisateurs peuvent écrire leurs propres programmes, dans un langage de programmation. Ces programmes peuvent être mis dans des lots, voir (Breheim, 1961) pour un exemple d'*open shop* sous le système d'exploitation FMS qui organise les tâches en lots.

pour l'IBM 709 (SOS, 1959) ou le RAND-SHARE Operating System pour l'IBM 7090 (1962). Dans le système d'Owen Mock, des programmes étaient mis dans un lot dont la durée d'exécution était estimée à une heure à peu près. Dans le GM/NAA Monitor, le traitement devenait plus complexe et se faisait en trois phases. D'abord une traduction pour convertir les données décimales en binaire et les programmes en instructions machine ; puis l'exécution du programme sous le contrôle du programmeur ; et finalement la sortie des résultats, imprimés ou perforés, après traduction[14]. L'évolution vers des moniteurs de plus en plus complexes sera poursuivie et trouvera son apogée en IBSYS (1962-1965) qui comme « *moniteur de moniteurs* […] *contient plusieurs des anciens systèmes* » (Hassitt, 1967, p. 24).

Toutefois, même si dans les histoires classiques des systèmes d'exploitation le type « traitement par lots » et la philosophie opérationnelle dominent le récit avant l'arrivée des systèmes à temps partagé, ce ne sont pas les seuls systèmes qui existent avant 1964. Une autre philosophie, très influente, est par exemple portée par le Comprehensive System of Service Routines (CSSR) du M.I.T. pour leur ordinateur Whirlwind (Bennington & Gaudette, 1956). Ce système, en développement continu depuis 1953, avait l'ambition de faciliter de manière générale l'accès à plusieurs groupes de programmes et d'automatiser certaines tâches du programmeur humain. Il y avait des groupes de programmes de lecture et écriture, de traduction et conversion, de diagnostic et de *debugging*, de contrôle et de moniteur, etc. L'idée derrière le *comprehensive system* est généraliste et inclut à la fois le(s) système(s) de programmation, le système d'exploitation et la bibliothèque de routines. Cette philosophie correspond à une approche plus unitaire et interconnectée d'organiser les logiciels autour de la machine.

On peut appeler cette philosophie « intégrative », d'après le terme *integrated system* souvent utilisé dans les années 1950 et 1960. Le terme est particulièrement populaire chez les personnes qui viennent du projet Whirlwind ou de Ramo-Woolridge. Dans le *Handbook for Automation, Computation and Control* (1959), le système intégré est défini comme « *l'interconnexion de quelques ou tous les programmes de service dans une unité organisée, contrôlée par le programmeur et semi-automatique ou complètement automatisée* »[15], ou encore : « *tous les programmes sont inté-*

14 « *an input-translation phase which converted data from decimal to binary, and programs from source to object language ; an execution phase which was almost exclusively under the programmer's direct control ; and an output translation phase which processed line printer output, punched card output (both decimal and binary), and accounting records* » (Patrick, 1987, p. 802).

15 « *Interconnection of some or all these different utility programs into an organized, programmer-controlled, semiautomatic or automatic whole is usually called an integrated system* » (Grabbe & *al.*, 1959, p. 184).

grés pour faire un seul système de calcul afin d'exclure l'utilisation de la machine par des sous-systèmes isolés »[16]. Des exemples de ce genre de système sont le CSSR du M.I.T., *MAGIC (Michigan Automatic General Integrated Computation)* de l'université de Michigan ou le système intégré développé à Ramo-Woolridge par le groupe de W.F. Bauer pour l'ordinateur ERA 1103 (1955).

Dans la philosophie des systèmes intégrés, tout est organisé autour de l'interconnexion entre programmes, ce qui contraste avec la philosophie opérationnelle qui sépare les logiciels aidant le programmeur des logiciels utilisés par l'opérateur. Aussi le(s) programme(s) qui contrôle(nt) la séquence des opérations n'ont pas nécessairement un statut à part. La bibliothèque des routines, en particulier les routines de service (*utility programs*), est le noyau du système. Le système d'exploitation du type intégré est surtout conçu comme structure d'accès à la machine par des groupes de routines qu'on peut combiner, varier et développer. C'est un aspect qui est également présent dans les systèmes d'exploitation modernes[17]. À cause de cette vision, ce type de systèmes avait tendance à ne pas toujours utiliser le cycle linéaire « *charger, assembler, compiler et exécuter* » (load, assemble, compile and execute) et à s'appuyer sur des routines interprétatives. Contrairement aux langages de programmation dans lesquels tout un programme est d'abord compilé pour être exécuté, dans un système interprétatif une ligne d'un programme est interprétée et directement exécutée. Ce système rend possible le remplacement du code de la machine par un langage plus avancé de sous-routines, et au lieu d'avoir plusieurs « *instructions de saut à une sous-routine* », on a « *une seule sous-routine qui interprète et supervise toutes les sous-routines appelées* »[18]. Sur beaucoup d'ordinateurs des années 1950, on disposait de systèmes interprétatifs pour faire des calculs en virgule flottante. Sur quelques ordinateurs avec beaucoup de mémoire de travail et des outils d'entrée et de sortie comme un *flexowriter* ou un écran cathodique, les systèmes interprétatifs préfiguraient parfois les systèmes interactifs qui naîtront avec le temps partagé. C'est le cas notamment pour le TX-0 ou le TX-2 développé par le Lincoln Lab au M.I.T.

16 « *The important concept here is that all items are integrated together to form one computation system to the exclusion of the use of the machine with isolated subsystems* » (Bauer, 1956, p. 8).

17 Cf. la remarque de G.H. Mealy : « *Many functions now classed as OS functions were first embodied as utility subroutines and programs* […] *Today, the library is an integral part of the OS – to the extent, for instance, that many programmers identify the UNIX system with its library rather than with its nucleus and shells* » (Mealy, 1987, p. 781).

18 « *The jump instructions in the main program which formerly directed control to the subroutines are eliminated* […] [the subroutines] *are all welded into one, an interpretive subroutine, which includes also a section to supervise the sequence in which the various operations are performed* », autrement dit « *the instruction code of the machine is not merely augmented, it is entirely replaced* » (Adams, 1954, pp. 16-3).

Aujourd'hui, on a du mal à distinguer clairement entre cette vision d'un système intégré et les systèmes de programmation. En général, il est, avant 1964, souvent difficile de bien distinguer entre ce qui relève d'un système de programmation ou d'un système d'exploitation. En effet, du point de vue de la philosophie intégrative, les routines de supervision, de moniteur ou de contrôle font, elles aussi, simplement partie de la bibliothèque des routines disponibles. Dans l'autre point de vue, les routines de moniteur supervisent et contrôlent les systèmes de programmation, elles sont donc hiérarchiquement au-dessus du système de programmation.

Même si la plupart des systèmes entre 1954 et 1964 peuvent être rangés dans les catégories de la philosophie opérationnelle ou de la philosophie intégrative, il existait aussi, comme le rappelle W.F. Bauer, « *un nombre de systèmes spécialisés, en particulier des systèmes de commande et contrôle qui utilisaient des idées avancées en systèmes d'exploitation* »[19]. Le système le plus connu et le plus influent est le système SAGE (*Semi Automatic Ground Environment*). SAGE était un projet majeur financé par l'armée américaine pour développer un système d'ordinateurs connectés par des lignes téléphoniques qui devait coordonner les informations venant des radars et des sites militaires afin d'obtenir une vue d'ensemble des activités aériennes. Le système devrait rassembler en temps réel toutes les informations nécessaires pour prendre des décisions militaires en cas d'une attaque (possiblement atomique) de l'URSS. Le Lincoln Lab du M.I.T. faisait partie du projet avec ses ordinateurs Whirlwind et TX-2. IBM a développé des ordinateurs AN-FSQ7 pour le projet. Plusieurs programmeurs de la RAND Corporation ont fondé la première société informatique du logiciel, SDC (*System Development Corporation*), pour écrire les programmes nécessaires pour le projet SAGE. Le projet a donc stimulé des innovations technologiques et en particulier en logiciel. Les systèmes développés par le M.I.T. et IBM contenaient des nouveautés, comme par exemple un sous-système interactif entre l'utilisateur et une visualisation de données sur un écran cathodique ; ou encore un sous-système de *teleprocessing*, l'utilisation des lignes téléphoniques pour transférer des données d'un ordinateur (ou périphérique) à un autre. D'une certaine manière, on pourrait appeler ces systèmes des systèmes distribués parce qu'un (ou plusieurs) processeur(s) central(ux) est couplé avec une variété de périphériques et qu'une communication en temps réel entre toutes ces unités est établie. Sur ces systèmes, beaucoup d'idées ont été développées et testées qui seront plus tard précieuses pour le développement des systèmes de multiprogrammation. Mais, à l'époque, ces systèmes étaient définis d'abord par leur caractère temps réel (parfois aussi appelé « accès direct » ou « en ligne »).

19 « […] *a number of special purpose systems, particularly command and control systems that utilized advanced operation system ideas ahead of their time* » (Bauer, 1972, p. 999).

Il y a eu d'autres systèmes spécialisés sans rapport avec le projet SAGE. Outre des applications militaires spécialisées, il y a eu surtout le développement de systèmes pour gérer en temps réel des usines. On citera dans ce domaine Thompson-Ramo-Woolridge et son RW-400, ainsi que General Electric qui avait créé son système GARDE pour l'ordinateur GE-312 spécialement pour le contrôle des générateurs d'électricité (1959). IBM aussi, utilisant l'expérience du projet SAGE, proposait des systèmes commerciaux de *teleprocessing* en temps réel. Le plus connu est le système SABRE pour la réservation de billets d'avion, mais il y a eu également des systèmes pour les sociétés des chemins de fer (TOPS) et d'autres encore.

Quand en 1966 IBM présente son système d'exploitation OS/360, on appellera ce système « de deuxième génération ». La première génération était composée des systèmes de traitement par lots, la seconde de l'OS/360 qui combinait l'idée de traitement par lots avec le caractère temps réel[20]. En effet, OS/360 combinait les deux lignes de développement au sein d'IBM : d'une part le traitement par lots né dans la communauté SHARE, d'autre part les systèmes en temps réel et de *teleprocessing* qu'IBM avait d'abord développé pour le projet SAGE avant de le commercialiser.

> Avec l'extension de l'utilisation des ordinateurs en télécommunications, la fonction de programme de supervision s'est élargie pour servir de pont entre traitement par lots et service à distance. Le programme de supervision qui en a résulté peut être utilisé pour contrôler un système qui traite des programmes en lots, ou un système qui contrôle plusieurs dispositifs de télécommunications, ou toute combinaison des deux[21].

20 « *Then, as now, the operating system aimed at non-stop operation over a span of many jobs and provided a computer-accessible library of utility programs. A number of operating systems came into use during the last half of the decade. In that all were oriented toward overlapped setup in a sequentially executed job batch, they may be termed 'first generation' operating systems. A significant characteristic of batched-job operation has been that each job has, more or less, the entire machine to itself, save for the part of the system permanently resident in main storage. During the above-mentioned period of time, a number of large systems-typified by SAGE, MERCURY, and SABRE-were developed along other lines; these required total dedication of machine resources to the requirements of one 'real-time' application. By and large, however, these real-time systems bore little resemblance to the first generation of operating systems, either from the point of view of intended application or system structure. Because the basic structure of OS/360 is equally applicable to batched-job and real-time applications, it may be viewed as one of the first instances of a 'second-generation' operating system. The new objective of such a system is to accommodate an environment of diverse applications and operating modes* » (Mealy, 1966).

21 « […] *as the use of computers extended into telecommunications, the function of the supervisory control program broadened to serve as the bridge between batch processing and service to remote locations. The resulting supervisory control program can be used to control a system which processes batch programs only, or a system dedicated to the control of telecommunications devices, or any combination of these two* » (Dines, 1966).

L'invention de l'*operating system* chez IBM

Bien que le terme *operating system* soit aujourd'hui prévalent dans la majorité des langues, plusieurs termes étaient en usage dans les années 1950 et 1960. Orchard-Hays dans sa synthèse sur les systèmes de programmation et d'exploitation de 1961 remarque :

> De nombreux termes sont utilisés pour les parties d'un système d'exploitation. Le mot « programme superviseur » est déjà apparu [antérieurement dans le texte]. Le superviseur est un programme qui garde le contrôle de la machine à tout moment et auquel retourne le contrôle après qu'une routine se termine ou s'arrête de manière inattendue. Les mots utilisés plus ou moins comme synonymes de « superviseur » sont : « routine exécutive », « moniteur », « routine du contrôle maître[22].

Plusieurs variantes de ces noms existent, telles que « routine de contrôle des séquences », « contrôle exécutif », etc. Les noms de ces routines désignent souvent l'ensemble du système[23]. On parlait de « système exécutif », « système superviseur », « système de contrôle », « système à séquencer des programmes » etc., et rarement de « système d'exploitation » (*operating system*).

Alors comment *operating system* est-il devenu le terme usuel ? Le terme semble avoir ses origines dans la communauté SHARE des utilisateurs des machines IBM. La première utilisation remonte à 1959 dans le contexte de développement du SHARE Operating System (SOS) piloté par un comité de SHARE et aidé par IBM. Dans les publications officielles dans le journal de l'ACM, le terme *operating system* n'est pas utilisé pour décrire le système de SHARE, ils parlent plutôt de SHARE 709 System[24]. Comme le note D.L. Shell qui présidait le comité de développement, « *le problème initial pour le comité était la définition même de ce que veut dire 'système'* »[25]. La partie du système qui contrôle les autres programmes est appelée *supervisory control program*, qui « *coordonne les parties du système SHARE 709 et est responsable du fait que l'ordinateur travaille de manière ininterrompue pendant le traitement d'un groupe de tâches indépendantes* »[26]. En

22 « *A number of terms have come into use for parts of an operating system. The term `supervisory program' has already appeared above. The supervisor is the program which maintains ultimate control of the machine at all times and to which control reverts when a routine finishes its function or when an unexpected stop or 'trap' occurs. Terms which are used more or less synonymously with 'supervisor' are 'executive routine,' 'monitor,' 'master control routine'* » (Orchard-Hays, 1961, p. 290).

23 L'utilisation d'une partie pour désigner l'ensemble est connue en linguistique comme *pars pro toto* ou métonymie.

24 709 renvoie à la machine IBM 709 pour laquelle le système fut développé.

25 « *The initial problem facing the committee was to define what was meant by a 'system'* » (Shell, 1959, p. 124).

26 « [it] *coordinates the use of the various parts of the SHARE 709 System and is responsible for maintaining the computer in continuous operation during the processing of a group of independent jobs* » (Bratman & Boldt, 1959, p. 152).

outre, le système a un format standardisé pour les tâches que la machine peut facilement interpréter, cette standardisation réduisant sérieusement le temps perdu entre deux tâches. C'est une description qui correspond parfaitement à un système de traitement par lots, mais ni le mot « lot » ni le mot « système d'exploitation » ne sont utilisés.

Dans le manuel de la communauté SHARE par contre, le système est appelé SOS, pour *SHARE Operating System*. Dans l'introduction du manuel, on peut lire :

> […] le SHARE Operating System, communément appelé SOS, est un complexe très flexible de langages, procédures et codes machine. Le système a trois objectifs : aider le programmeur du 709 pendant la préparation des programmes ; alléger l'opérateur des tâches qui peuvent être automatisées ; et rendre l'installation informatique plus efficace et mieux documentée[27].

SOS est en réalité « un système intégré » qui contient trois sous-systèmes : le compilateur-assembler-traducteur de SHARE (SCAT) ; le système pour le *debugging* ; et le moniteur.

La nomenclature utilisée dans les publications officielles (ACM) et celle utilisée dans la pratique (la communauté SHARE) ne coïncident pas. Pourtant, il est évident que le terme *operating system* ne s'était pas encore imposé et que la définition même était encore flottante, désignant parfois le système de programmation ou même le système intégré. Cela changera avec les systèmes ultérieurs à SOS.

Le SHARE Operating System aura peu de succès dans la communauté SHARE parce que FORTRAN n'était pas bien supporté et que son langage d'instructions était trop compliqué (Akera 2001, pp. 731-733). En 1961, 76 % des installations IBM sont opérées sous le système Fortran Monitor System (FMS) au lieu de SOS (Larner 1987, p. 819). FMS avait été développé par un grand utilisateur institutionnel North American Aviation en 1959 pour faciliter l'usage de FORTRAN. FMS était « *un moniteur, un programme superviseur pour 709/7090 FORTRAN, FAP*[28] *et les programmes objet* »[29]. Si SOS avait été conçu comme un outil de programmation, FMS était dès le début un chargeur et connecteur (*loader & linker*) pour des programmes en FORTRAN. Peut-être que le succès de FORTRAN et son profil de « *langage*

27 « […] *the SHARE operating system, familiarly known as SOS, is a highly flexible complex of languages, procedures and machine codes. The threefold purpose of the System is to provide assistance to the 709 programmer in the coding and check-out phase of program preparation, to assume from the 709 machine operator those burdens that may be sensibly automated and to provide the computer installation with an efficient operation and complete and accurate records on machine usage* » (Homan and Swindle, 1959, sec. 01.01.01).

28 FAP est le macro-assembleur du IBM 709/7090.

29 « […] *the Monitor is a supervisory program for 709/7090 FORTRAN, FAP, and object programs* » (Reference Guide, 1961, p. 61).

de programmation »[30] a rendu lentement possible la distinction claire entre le système de programmation et le système d'exploitation (même si cette séparation est parfois artificielle et problématique). Par l'intégration du langage FORTRAN dans un environnement comme FMS qui facilite l'écriture, la compilation et la combinaison de programmes, une distinction entre langage et système (d'exploitation) devient visible pour l'utilisateur.

Cette évolution est explicitée par George H. Mealy qui faisait partie du groupe de la RAND ayant amélioré SOS pour intégrer FORTRAN, le résultat étant le RAND-SHARE Operating System (1962). Dans son rapport sur les *Operating Systems*, Mealy écrivait :

> On a une machine pour exécuter des tâches, pas pour programmer des systèmes. On appelle souvent les systèmes entre le programmeur et la machine des « systèmes de programmation », mais cela met trop en exergue les auxiliaires mécaniques de codage et ne met pas en valeur d'autres aspects de l'opération. Par *operating systems* on désigne l'ensemble des outils de programmation, debugging et outils opérationnels avec lequel travaille le programmeur[31].

Ainsi, *operating system* commence à englober et à contrôler de plus en plus les systèmes de programmation de l'ordinateur. Dans la même ligne, Bob Bemer d'IBM considérait les *operating systems* comme la Phase III dans le développement des systèmes de programmation[32], l'*operating system* englobant littéralement les systèmes de programmation (Bemer 1962).

Une séparation semblable entre système de programmation et *operating system* se trouve dans l'introduction de manuel du RAND-SHARE Operating System :

> Un *operating system* est un complexe de routines d'ordinateur qui sont utilisées pour faire entrer et sortir les programmes et les données de la machine, pour transformer des données (incluant la compilation et l'assemblage de programmes), superviser des tâches et les mettre en

30 On parle d'abord plutôt de systèmes de programmation, le fait de parler de *langage* de programmation est apparemment né dans les communautés des utilisateurs, en particulier USE (1955). L'emphase sur le langage a probablement aidé à fortifier l'idée qu'un système de programmation peut être développé indépendamment d'une machine et peut être d'application universelle, voir (Nofre, Priestley & Alberts, 2014). FORTRAN par exemple est un système qui comporte deux éléments, le langage et le traducteur qui adapte le code à la machine.

31 « *The object of having a machine is to run jobs, not programming systems. To call the systems that stand between the programmer and the machine « programming systems » is to place undue emphasis on mechanical coding aids and not enough emphasis on the other aspects of operation. By 'operating systems' we shall mean the whole complex of programming, debugging and operational aids with which the programmer deals* » (Mealy, 1962, p. 4).

32 Pour Bemer, la Phase I était la programmation de l'ordinateur sans aide, la Phase II la programmation partiellement automatisée (conversion, chargement, connexions, assemblage et compilation), et en Phase III finalement, le système d'exploitation prend en charge plusieurs tâches comme par exemple la communication avec les périphériques entrée et sortie, etc.

séquence, et pour faciliter la communication entre le programmeur et les éléments du système d'exploitation[33].

Comme pour SOS, le RAND-SHARE Operating System vise trois choses : l'efficacité dans l'utilisation du temps de la machine ; l'efficacité dans l'opération de la machine ; et l'efficacité dans la programmation. Sauf que, désormais, l'opérateur n'est même plus mentionné, et l'*operating system* commence à gouverner les systèmes de programmation. Cette évolution remplace l'opérateur par l'*operating system* comme interface principale entre programmeur et machine (du moins, en théorie, mais certainement pas dans la pratique !).

Cette tendance s'approfondit avec l'*operating system* développé par IBM, IBSYS.

Le système d'exploitation 7090/7094 IBSYS est constitué d'un ensemble intégré de programmes-systèmes qui opèrent sous le contrôle et la coordination du *System Monitor*. Le *System Monitor*, en coordonnant l'opération des sous-systèmes, permet qu'une séquence de tâches sans relation entre elles peut être exécutée sans, ou presque sans, l'intervention d'un opérateur. En réduisant le degré de participation humaine dans la mécanique du traitement d'information, le 7090/7094 IBSYS Operating System garantit que les tâches sont traitées plus vite, plus efficacement et avec moins d'erreurs humaines[34].

Cette description efface littéralement l'opérateur de l'équation et met l'*operating system* à sa place, comme catalyseur pour le programmeur. Le système contrôle et réduit les problèmes qui peuvent apparaître en traitement d'informations et qui sont dus à l'homme. L'*operating system*, en particulier le moniteur, est mis en tête de la hiérarchie des programmes et contrôle la configuration de l'ordinateur, ses périphériques et ses utilisateurs. Et avec le nom même d'*operating system*, la philosophie opérationnelle s'impose de plus en plus.

33 « *An operating system is a complex of computer routines which are used to get programs and data into and out of the machine, transform data (including program assembly and compilation), supervise job and task sequencing, and facilitate the communication between the programmer and components of the operating system* » (Bryan, 1962, p. III).

34 « *The 7090/7094 IBSYS Operating System consists of an integrated set of system programs operating under the executive control and coordination of the System Monitor. The System Monitor, by coordinating the operation of the subsystems, allows a series of unrelated jobs to be processed with little or no operator intervention. By reducing the degree of human participation in the mechanics of data processing, the 7090/7094 IBSYS Operating System ensures that jobs are processed faster, more efficiently, and with less likelihood of human error* » (IBSYS, 1964, p. 3).

Conclusion

Le développement fulgurant du logiciel va de pair avec les avancées technologiques en matière de mémoire de stockage et de vitesse de processeur. Le développement des programmes est d'abord porté par les utilisateurs, mais vers 1960 une nouvelle industrie, celle du logiciel, naît. Avec la croissance en quantité et en complexité des programmes, il devient nécessaire qu'un ordinateur soit accompagné d'une sorte de système d'exploitation. Depuis la fin des années 1950, l'utilisateur humain qui calculait déjà plus lentement que l'ordinateur, est aussi dépassé par les périphériques qui lisent et écrivent des données beaucoup plus vite que lui. Du lecteur de cartes perforées fonctionnant à deux ou trois cartes par seconde, on passe à des bandes magnétiques dont la vitesse de lecture atteint plus de 10 000 cartes par seconde. En outre, avec FORTRAN, l'ordinateur « écrit » (c'est-à-dire qu'il assemble, compile et exécute) un programme plus vite qu'un utilisateur humain, une fois que l'*automatic programming* ou *programming programs* sont devenus communs. Tous ces éléments conduisent à la naissance des systèmes d'exploitation.

Plusieurs philosophies coexistent, pour ne pas dire que chaque ordinateur ou chaque entreprise suit sa propre philosophie, incarnée dans son propre système d'exploitation. Toutefois, une philosophie emporte la majorité des suffrages, la philosophie opérationnelle sous-entendue avec le terme *operating system*. Cette vision s'appuie sur l'automatisation de l'opérateur humain et est étroitement liée au traitement par lots (*batch processing*). Elle est surtout portée par IBM qui domine le marché des ordinateurs. En quelque sorte, cette philosophie est la poursuite de la stratégie commerciale IBM consistant à louer les ordinateurs et à séparer l'utilisateur de la machine par l'intermédiaire des opérateurs, qui, sous forme automatisée, deviennent l'*operating system*[35].

Avec l'avènement de la multiprogrammation et du *time-sharing* la variété et la complexité des systèmes d'exploitation augmentera énormément. Même si désormais beaucoup de systèmes vont assumer des fonctionnalités qui sont étrangères à l'ancien opérateur humain, comme la segmentation, l'ordonnancement etc., le terme *operating system* va rester et devenir le terme utilisé par tout le monde.

35 Évidemment, la pratique ne correspond pas complètement à cette vision automatisée qui repose souvent sur des métaphores. L'opérateur humain reste nécessaire, mais son rôle change : il devient plutôt assistant ou administrateur du système d'exploitation.

Bibliographie

Adams C.W., Gill, S. & *al.* (eds.) (1954). *Digital Computers: Business Applications.* Summer program.

Adams C.W. (1955). « Developments in programming research ». *AIEE-IRE 1955 Eastern joint AIEE-IRE computer conference proceedings*, pp. 75-79.

Adams C.W. (1987). « A batch-processing operating system for the Whirlwind I computer ». *AFIPS Conference Proceedings,* vol. 56, pp. 785-789.

Akera, A. (2001). « Voluntarism and the fruits of collaboration: The IBM user group Share ». *Technology and Culture*, 42(4), pp. 710-736.

Bauer W.F. (1956). « An Integrated Computation System for the ERA-1103 ». *ACM,* 3 (3), pp. 181-185.

Bauer W.F. & West G.P. (1957). « A system for general-purpose digital-analog computation ». *ACM*, 4 (1), pp. 12-17.

Bauer W.F. (1956). « Use of Automatic Programming ». *Computers and Automation*, 5 (11), pp. 6-11.

Bauer W.F. (1958). « Computer Design from the Programmer's Viewpoint ». *Proceedings Eastern Joint Computer Conference*, December 1958, pp. 46-51.

Bauer W.F. & Rosenberg A.M. (1972). « Software – Historical perspectives and current trends ». *Fall Joint Computer Conference,* 1972, pp. 993-1007.

Bemer R. (1962). « The Present Status, Achievement and Trends of Programming for Commercial Data Processing ». *In* Hoffmann (dir.) *Digitale Informationswandler*, Wiesbaden, pp. 312-349.

Bennington H.D. & Gaudette C.H. (1956). « Lincoln Laboratory Utility Program System ». *AIEE-IRE 1956 joint ACM-AIEE-IRE western computer conference*, p. 21.

Bratman H. & Boldt I.V. (1959). « The SHARE 709 System: Supervisory Control ». *Communications of the ACM*, 6 (2), pp. 152-155.

Breheim D.J. (1961). « 'Open Shop' Programming at Rocketdyne Speeds Research and Production ». *Computers and Automation,* 10 (7), pp. 8-9.

Brinch Hansen P. (2001). *Classic Operating Systems: From Batch Processing to Distributed Systems.* Berlin : Springer.

Bryan G.E. (1962). *The RAND Share operating system manual for the IBM 7090.* Memorandum RM-3327-PR, Santa Monica, CA.

Campbell-Kelly M. (2003). *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry.* Cambridge, MA : MIT Press.

Ceruzzi P. (2003). *A history of modern computing* (2nd ed.). MIT Press, Cambridge, Massachussetts.

Culler G.J. & Fried, B.D. (1963). *An Online Computing Center for Scientific Problems*. M19-3U3, TRW report.

Dines R.S. (1966). « Telecommunications and supervisory control programs ». *Computers and Automation* 15 (5), pp. 22-24.

Drummond R.E. (1987). « BESYS revisited ». *AFIPS Conference Proceedings,* vol. 56, pp. 805-814.

Fisher F.P. & Swindle G.F. (1964). *Computer programming systems.* New York : Holt, Rinehart and Winston.

Frank W.L. (1956). « Organization of a Program Library for a Digital Computer

Center ». *Computers and Automation,* 5 (3), pp. 6-8.

Grabbe E.N., Ramo S. & Woolridge D.E. (1959). *Handbook of Automation, Computation and Control,* volume II. New York : Wiley.

Hassitt, A. (1967). *Programming and Computer systems.* New York & London : Academic Press.

Homan C.E. & Swindle G.F. (1959). *Programmer's Manual for the SHARE Operating system*. IBM.

Hopper G. (1954). *ACM Glossary.* ACM.

Reference Guide to the 709/7090 FORTRAN Programming System (1961) (includes material from IBM 709/7090 FORTRAN Monitor, form C28-6065). IBM : Poughkeepsie.

IBM 7090/7094 IBSYS system operator's guide (1964). Poughkeepsie : IBM.

Knuth D.E. & Pardo L. (1979) « The early development of programming languages ». *In* J. Belzer, A.G. Holzman & A. Kent (dir.). *Encyclopedia of Computer Science and Technology.* Marcel Dekker : New York, pp. 419-496.

Krakowiak S. (2014). « Les débuts d'une approche scientifique des systèmes d'exploitation ». *Interstices,* Février.

Krakowiak S. & Mossière J. (2013). « La naissance des systèmes d'exploitation ». *Interstices,* Avril.

Larner R.A. (1987). « FMS: The IBM FORTRAN Monitor System ». *AFIPS Conference Proceedings,* vol. 56, pp. 815-820.

Mealy G.H. (1962). *Operating Systems.* RAND Report P-2584. Republié partiellement in Rosen 1967.

Clark W.A., Mealy G.H. & Witt B.I (1966). « The functional structure of OS/360 ». *IBM Systems Journal*, 5 (1), pp. 3-51.

Mealy G.H. (1987). « Some threads in the development of early operating systems ». *AFIPS Conference Proceedings,* vol. 56, pp. 779-784.

Mock O.R. (1987). « The North American 701 Monitor ». *AFIPS Conference Proceedings,* vol. 56, pp. 791-795.

Moncreiff B. (1956). « An automatic supervisor for the IBM 702 ». *AIEE-IRE 1956 Joint ACM-AIEE-IRE western computer conference*, pp. 21-25.

Nelson E. (1969). « Computer Installation at TRW systems – Some Experiences and Lessons ». *Computers and Automation* 18 (8), pp. 21-22.

Nofre D., Priestley M. & Alberts G. (2014). « When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-1960 ». *Technology and Culture*, 55(1), pp. 40-75.

Orchard-Hays W. (1961). « The Evolution of Programming Systems ». *Proceedings of the IRE*, 49 (1), pp. 283-295.

Patrick, R.L. (1987). « General Motors/North American Monitor for the IBM 704 computer ». *AFIPS Conference Proceedings,* vol. 56, pp. 796-803.

Rosen S. (1967). *Programming Systems and Languages.* New York : McGraw-Hill.

Sackman H. (1970). *Man-Computer Problem solving.* Princeton etc. : Auerbach.

Shell D.L. (1959). « SHARE 709 system: a cooperative effort ». *Journal of the ACM*, 6 (2), pp. 123-127.

SHARE Operating System Manual, Distribution 1 to 5 (1960). Poughkeepsie, IBM.

Tanenbaum A. (2001). *Modern operating systems* (2nd ed.). Upper Saddle River, NJ : Prentice Hall.

La Saga des machines-langage et -système

François Anceau
Collaborateur bénévole, Lip 6, Sorbonne-Université/UPMC.

Résumé

Parallèlement à l'évolution des machines informatiques standard, la recherche et l'industrie informatique ont développé, dès le début des années 1960, des machines originales fondées sur l'idée de déplacer la frontière entre le logiciel et le matériel au profit de ce dernier. Cette évolution de l'architecture des machines s'appuyait sur l'évolution technologique (rapidité du matériel et possibilité de réaliser des machines complexes). Plusieurs nouvelles classes de machines en ont découlé. Nous retiendrons ici la classe des machines-langage orientées vers l'exécution directe de langages évolués et celle des machines-système possédant des mécanismes systèmes au niveau matériel et surtout accédant à ses divers objets via des descripteurs. Cette évolution promettait une simplification et une accélération des fonctions appartenant à l'origine aux couches logicielles. Ces espoirs n'ont pas été complètement satisfaits. Après un départ glorieux, les machines-langage ont disparu dès les années 2000, et les machines-système ont été simulées sur d'autres machines pour continuer à assurer leurs services. Seule, la lignée des machines-système x86 et amd64 a continué sa progression grâce à sa prééminence dans les ordinateurs personnels. Depuis, cette lignée se trouve en compétition avec les ARM venus du monde de l'enfoui et des portables.

Mots clés : architecture des ordinateurs, machines spécialisées, machines-langage, machines-système, architecture virtuelle, intérieur-décor.

Introduction

Dès le début des années 1960, les concepteurs de machines informatiques ont essayé de casser le schéma conceptuel dans lequel ils commençaient à s'enfermer. À cette époque, le traitement de l'information se résumait au calcul numérique. Ces calculs se présentaient alors sous deux formes très distinctes :

- les calculs simples se répétant sur un ensemble assez important de données : par exemple le calcul des feuilles de paie, la réalisation de tables (logarithmes, trigonométriques, éphémérides pour la navigation, tables de tir…), relevant du domaine de la mécanographie ;
- les calculs non répétitifs réalisés à la demande (rares et coûteux).

À cette époque, il était beaucoup plus simple de consulter une table, que de calculer un logarithme, ou une fonction trigonométrique, compte tenu de la lenteur, et de la rareté de ces machines à calculer.

L'électronique était alors en pleine mutation grâce à l'arrivée des transistors au silicium dans le monde de l'électronique industrielle en remplacement des tubes à vide. Outre une diminution de taille et de poids, l'arrivée de ces composants allait transformer l'utilisation des machines informatiques par l'augmentation importante de leur fiabilité, faisant passer de la journée à la semaine leur temps moyen de fonctionnement entre pannes, permettant ainsi les longues exécutions des programmes devenus plus complexes.

Le passage du calcul numérique au traitement de l'information, se déroula pendant les années 1960. Cette extension du domaine d'application de ce genre de machine s'appuyait sur des travaux théoriques montrant la généralité de la notion de calcul :

- la thèse de Church-Turing qui montre que tout calcul peut se faire par la combinaison de quelques opérations simples, ce qui a permis de définir les limites des machines à calculer programmables ;
- la diagonalisation de Gödel montre que toute transformation d'ensembles énumérables peut se coder comme un calcul ;
- le théorème de programmation structurée de Corrado Böhm et Giuseppe Jacopini (1966) a ouvert la porte de la programmation moderne.

Toutefois l'unification entre le calcul et le traitement de l'information était pressentie depuis longtemps. G. W. Leibniz, avec sa « Caractéristique Universelle », avait déjà compris en 1666 la possibilité d'utiliser le calcul logique pour autre chose que des nombres. Ada Lovelace avait aussi prédit cette évolution en 1842 dans ses commentaires de sa traduction de la description de la machine de Babbage (Tools, 1992). Toutefois, sa mise en œuvre sera progressive pour n'être effective que vers 1965, provo-

quant la naissance conjointe de l'informatique et des ordinateurs (Mounier-Kuhn, 2010 ; Anceau, 2000, p. 89). En effet, beaucoup de ces machines à calculer ne pouvaient traiter que des chiffres par des codages plus ou moins optimisés pour le calcul (par exemple le code bi-quinaire) ce qui rendait la manipulation de lettres ou d'autres items particulièrement difficile. L'arrivée de l'informatique et des ordinateurs est marquée par la possibilité de manipuler individuellement des éléments binaires regroupés dans des mots.

Machines spécialisées

Très tôt, l'idée de spécialiser les machines informatiques afin de les optimiser pour des tâches précises inspira les concepteurs. De nombreuses voies s'ouvraient alors, allant de l'adaptation de la structure des machines de base à un type d'application, jusqu'au remplacement des langages d'instruction par des langages de programmation préexistants. Dans ce texte, nous ne traiterons que des processeurs et coprocesseurs-langage et des processeurs-système :

- les processeurs et coprocesseurs-langage exécutent directement les langages intermédiaires des langages de programmation ;
- les processeurs-système exécutent directement des fonctions systèmes élaborées, comme l'adressage segmenté, la gestion des tâches, leur synchronisation et l'isolement hiérarchique par des anneaux de protection entre les couches du système et celles des utilisateurs.

La Figure 1 donne un aperçu de la grande variété des processeurs spécialisés.

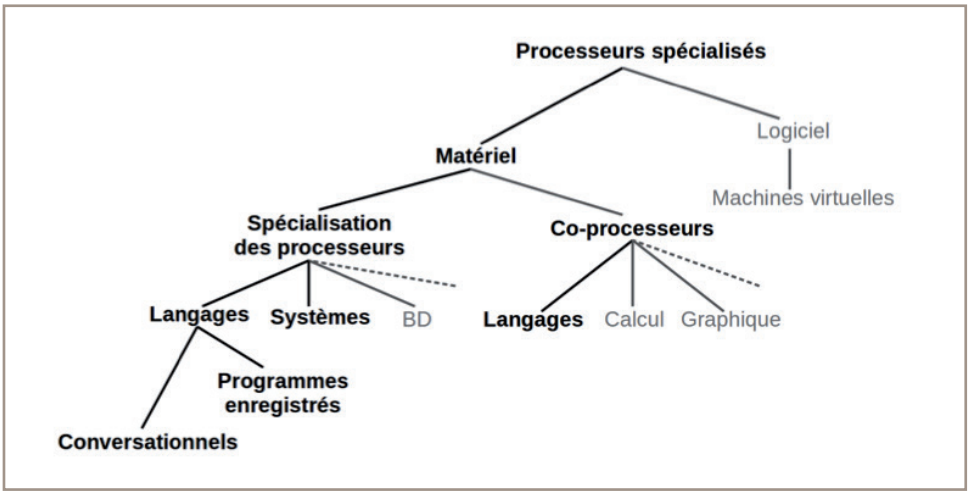

Figure 1 – Famille des processeurs spécialisés

Machines-langage

Le début des années 1960 vit la naissance des premiers langages informatiques capables de décrire des processus de traitement de l'information. Dès le départ, ceux-ci se divisèrent en plusieurs classes :

- les langages pour le calcul répétitif sur des enregistrements (par exemple Cobol) ;
- les langages pour le calcul scientifique (par exemple Fortran) ;
- les langages pour la recherche (par exemple : Lisp, IPL5 (pour la programmation fonctionnelle), Algol (pour un usage général).

Le langage Cobol fut conçu dans le prolongement des travaux mécanographiques qui effectuaient des traitements simples se répétant sur une multitude d'enregistrements à l'image des cartes perforées. L'importance de traitements plus complexes et non répétitifs (mais comportant des phases itératives !) se fit sentir dans les nouveaux langages comme Fortran, Algol, Lisp, etc. qui jetèrent les bases des langages de programmation modernes.

La programmation au niveau élémentaire d'un ordinateur (langage dit d'assemblage) a d'abord été vue comme l'utilisation de commandes contingentes au matériel utilisé. On a vite compris que l'ordinateur exécutait un algorithme interprétant ses instructions et qu'il fallait relativement peu d'efforts pour qu'il devienne universel, c'est-à-dire capable d'exécuter un large ensemble d'algorithmes, dont ceux issus des couches basses du logiciel. L'idée est donc venue d'utiliser le matériel d'une machine informatique pour exécuter le plus directement possible les langages évolués dans le but d'en faire les langages de base des nouvelles machines.

À ce niveau, il faut distinguer deux types de langages informatiques :

- les langages conversationnels (par exemple : Lisp, Basic, APL, Python, TRAC…) pour lesquels la machine répond directement, comme une calculette, à des commandes frappées sur un clavier ;
- les langages enregistrés (par exemple : Algol, Pascal, C, ADA, Java, Fortran, Cobol….) pour lesquels la machine traite, en plusieurs étapes successives des programmes préparés sous forme de fichiers.

Les langages conversationnels offrent généralement aussi la possibilité de grouper des commandes dans des fichiers. Certains langages conversationnels sont devenus des langages de commande pour l'exécution des travaux sur les ordinateurs (ex : Basic sur les premiers micro-ordinateurs et Python plus récemment).

Les langages enregistrés sont généralement traités en deux temps. Une première phase, appelée compilation transforme les programmes, écrits comme des textes, en des programmes écrits dans

des langages d'instructions spécifiques aux langages évolués, appelés langages intermédiaires par ex : P-Code pour Pascal, Bytecode pour Java… L'exécution de ces langages intermédiaires peut se faire de trois manières :

- par l'exécution du code intermédiaire par un interprète logiciel, généralement appelé « machine virtuelle logicielle » ;
- par la construction matérielle de machines physiques exécutant directement ces codes intermédiaires comme leurs langages d'instruction (voir le paragraphe suivant) ;
- par la traduction du code intermédiaire en instructions d'une machine cible ou dans un autre langage évolué qui devra être lui-même exécuté.

Les langages intermédiaires pour les langages enregistrés sont souvent de la forme post-fixée dans laquelle les opérateurs suivent les données (ex : A+B s'écrit AB+) qui présente la propriété de s'exécuter avec une pile, ou une file qui permet alors une exécution pipe-line (Anceau, Baille, & Schoellkopf, 1977).

Ordinateurs conversationnels

Plusieurs machines ont été construites pour exécuter directement les langages conversationnels. Compte tenu de la faible puissance de calcul demandée, il s'agit souvent de machines classiques munies d'un logiciel adéquat, appelé interprète.

• *Machines Lisp*

Plusieurs laboratoires se sont lancés dans l'étude de machines Lisp, par exemple :

- la machine CADR du MIT (1981) produite et commercialisée par Symbolics (LM-2, 3600…) ;
- les machines M3L et MAIA de l'Université Paul Sabatier de Toulouse ;
- le coprocesseur COLIBRI de Siemens ;

• *Machine Alvan*

Cette petite machine franco-américaine exécutait un langage voisin de celui du macro-générateur TRAC. Plusieurs machines ont été vendues à des entreprises pour des travaux de gestion.

• *Machines APL (IBM)*

- IBM 5100/5110 (portable).
- Extension microprogrammée de l'IBM 370/145.

• *Machines BASIC*

Pratiquement tous les micro-ordinateurs des années 1970 exécutaient le BASIC (*via* un interprète) à la fois comme langage de programmation et comme langage de commande.

Ordinateurs pour langages enregistrés

Plusieurs machines de puissance moyenne, ou forte, exécutaient directement le langage intermédiaire issu de la compilation d'un, ou de plusieurs, langages de programmation enregistrés. Il faut remarquer que cette approche suppose que l'ensemble du logiciel de base de ces machines doit être écrit dans le langage évolué (le système d'exploitation et le compilateur lui-même).

- *Machine Cobol*

Un prototype de machine Cobol a été développé à l'IRIA au début des années 1970 par R. J. Chevance (1974).

- *Machines Algol*

La société Burroughs Corporation a développé dans les années 1960 une ligne de machines puissantes (B5000 à B7700) (multiprocesseurs), exécutant un langage intermédiaire pour Algol / Cobol / Fortran (Organik, 1977). Ces machines ont connu une diffusion mondiale, avant de devoir s'effacer devant l'évolution des machines standards au début des années 1970.

- *Machines Pascal*

La société Western Digital a développé une série d'ordinateurs personnels appelés Pascaline (WD-9000/900/90, SB-1600) bâtis en utilisant un ensemble de circuits intégrés MCP-1600 constituant une sorte de mécano permettant la réalisation de cœurs de processeurs génériques.

Un prototype de machine Pascal, appelée PascHLL, a été étudié à l'Université de Grenoble à la fin des années 1970 pour démonter l'intérêt de l'exécution sur file (Schoellkopf, 1977 ; Anceau, Baille, & Schoellkopf, 1977).

- *Machine ADA*

La société Intel développa en 1981 un processeur bi-boitier appelé Intel iAPX 432 adapté à l'exécution du langage ADA et comportant des mécanismes de gestion mémoire du type de ceux existant dans les machines-système, ce qui classe ce processeur dans les deux familles. Très apprécié des universitaires, mais trop lent, il fut rapidement abandonné.

- *Machines JAVA*

Dans les années 1990, la société Sun a développé des microprocesseurs Java appelés Pico-Java (O'Connor & Tremblay, 1997) dans l'espoir du développement de stations Internet (*net-computers*). Malheureusement, l'évolution de la micro-informatique fut différente et ces microprocesseurs furent abandonnés.

Dans la première moitié des années 2000, un prototype de machine Java appelé JMQ pour Java Machine on Queue, de haute performance à exécution sur file a été étudié à l'UPMC (Palus, 2006).

Dans les années 2000, la société ARM a développé une extension du code de ses processeurs, appelée Jazelle, adaptée à l'exécution du Byte-code sur ses microprocesseurs.

Machines-système

La complexification des machines informatiques à la fin des années 1960 a permis d'effectuer un saut dans la nature des systèmes d'exploitation. De nouvelles fonctions ont été ajoutées dans de nombreux processeurs :

- la pagination a résolu le problème de la gestion automatique des mémoires principales (machine ATLAS de l'Université de Manchester en 1962) ;
- l'évolution de la vitesse des processeurs par l'utilisation d'architectures internes complexes (*pipe-line*, exécution dirigée par les données…) ;
- la transposition matérielle ou microprogrammée de fonctions systèmes (gestion de la mémoire segmentée, gestion des processus et des synchronisations, multitraitement, virtualisation, amélioration de la sécurité par des anneaux de protection), ce qui nécessite l'adjonction de nouvelles instructions très spécifiques.

La caractéristique discriminante des machines-système est l'utilisation de descripteurs utilisés par le matériel pour accéder aux différents objets manipulés (segments (données, programmes ou bibliothèques), tâches, entrées-sorties…). Ces descripteurs sont constitués de mots ou de double-mots qui contiennent généralement :

- la localisation des objets,
- leur nature (segment mémoire, tâche, entrées-sortie…),
- leur état (dans le cas de tâches : active, en attente de synchronisation…),
- leur propriétaire,
- leur niveau de protection (anneaux),
- leur type d'accès (dans le cas de segments mémoire : lecture, lecture-écriture),
- leur disponibilité (en mémoire vive ou secondaire),
- etc.

Machine Multics

En 1964, un consortium constitué du MIT, des laboratoires BELL, et de General Electric décida d'étudier un nouveau système d'exploitation, appelé Multics (Organik, 1972) fondé sur les nouveaux concepts qui émergeaient des universités et des constructeurs. Ce système est surtout connu pour :

- la gestion segmentée de la mémoire : les segments représentent une division fonctionnelle de l'espace de travail ;
- la gestion des processus et des synchronisations (en logiciel sur Multics) ;
- son écriture en langage évolué (PL1) ;

- l'utilisation semi-matérielle d'anneaux de protection ;
- la pagination (matérielle) servit à construire une énorme mémoire virtuelle englobant tous les fichiers et les accès aux périphériques.

Pour que les temps d'exécution de ces fonctions restent raisonnables, des extensions du matériel furent réalisées, surtout pour la segmentation. L'ordinateur utilisé, un GE635 de General Electric fut modifié par le MIT et rebaptisé GE645. Il servit de modèle à cette nouvelle lignée.

La complexité de Multics incita à la création d'un système plus simple appelé Unix qui fonctionnait sur des machines standards, en particulier les Vax de Digital Equipment et les machines SUN, qui n'avaient pas les caractéristiques des machines-système. Une version d'Unix appelée Linux a été adaptée aux processeurs x86-amd64 des ordinateurs personnels avec un grand succès commercial. Toutefois cette adaptation n'utilise pas la mémoire segmentée du x86. Elle la recouvre par des segments géants pour la transformer en mémoire plate c'est-à-dire non structurée.

Les prototypes

Les ordinateurs de la fin des années 1960 et du début des années 1970 bénéficièrent d'évolutions technologiques qui leur permirent d'accroître fortement leur complexité (circuits intégrés de faible complexité, microprogrammation et mémoires à tores). Ceci provoqua l'arrivée d'une floraison de machines-système prototypes issues de l'université et de l'industrie :

- le GE645 du projet Multics que nous avons déjà cité ;
- le MU5, de l'université de Manchester[1] ;
- le Level 64 d'Honeywell-Bull ;
- etc.

À la fin des années 1970 de nouveaux projets furent lancés :

- le Solar 16 de Télémécanique pour le temps réel qui possédait des instructions machines spécifiques pour la synchronisation des tâches;
- l'Intel iAPX 432 bi-circuits, (déjà cité dans les machines-langage) ;
- l'Intel 286, qui fut une première tentative d'évolution de la gamme x86 vers les machines-système.

Les machines de série

Plusieurs de ces prototypes débouchèrent sur des séries :

- le GE 645 donna naissance à l'Honeywell 8180, puis au Bull DPS 8000 Multics ;
- le MU5 donna naissance à la ligne des ICL 2900 ;

1 « MU5 », Manchester University [URL : http://www.cs.manchester.ac.uk/about-us/history/mu5/].

- le Level 64 donna naissance aux lignes des Bull-GE GECOS 7, Bull DPS 7/7000[2] et à celle des NEC ACOS 4 ;
- Télémécanique commercialisa la ligne des Solar 16 eut un grand succès commercial.

Cas des x86

Le microprocesseur Intel 386 et ses descendants sont l'exemple le plus remarquable de l'évolution d'une lignée de machines-système. L'Intel 286 ne fut utilisé que comme un super 8086 à cause d'erreurs de définitions qui empêchèrent de l'utiliser comme une machine-système. Par contre, l'Intel 386 fut un succès qui permit de développer des systèmes d'exploitation modernes comme les Windows, OS2 et de supporter Unix, Linux et Mac-OS. Le passage à 64 bit proposé par AMD supprime de fait la segmentation. Il est devenu le standard amd64 utilisé par tous les constructeurs de ce type de circuit.

Ces processeurs ont traversé plus de 30 ans d'évolutions de l'informatique. Leur passage aux multi-cœurs et l'utilisation de techniques d'exécution très avancées leur donne une position dominante dans tous les domaines de l'informatique, des postes de travail aux superordinateurs.

2 Cf. les textes sur le GCOS 7 sur le site de la Fédération des Équipes Bull, FEB-Patrimoine [URL : http://www.feb-patrimoine.com/projet/gcos7/gcos7.htm] et [URL : http://www.feb-patrimoine.com/projet/gcos7/gcos7-system-architecture.htm].

La saga des RISC

Le déplacement de fonctions du logiciel vers le matériel n'a pas eu que des avantages. D'abord, ces fonctions se sont trouvées de fait figées, tandis que les idées évoluaient rapidement. Ceci a parfois obligé des logiciels plus récents à contourner ces mécanismes pour retrouver leur liberté. Ensuite, les processeurs, surchargés par ces fonctions complexes devenaient plus lents, plus complexes, donc plus difficiles à produire et plus chers.

En 1975, John Cock, chercheur chez IBM, proposa de réaliser des machines plus simples (IBM 801), quasiment réduites à l'essentiel, appelées des RISC (pour Reducted Instruction Set Computers) (Étiemble, 1991). De telles machines se développèrent dans les années 1980. Outre un fort effet de mode, leur vitesse d'exécution s'est trouvée doublée, mais leurs programmes s'étant allongés compte tenu de la faible expressivité du code des premiers RISC compensaient cet avantage…

La mode des RISC rendit obsolète une partie des machines complexes, appelées CISC (pour Complex Instruction Set Computers) qui virent leur marché se réduire. À part les x86-amd64, les autres gammes de machines complexes disparurent ou furent réduites à des logiciels d'émulation sur PC.

Vers le milieu des années 1990 un nouveau mode d'exécution appelé *data-*

flow restreint vint toucher le monde des x86 et quelques RISC « assagis » par l'usage d'instructions complexes (comme les ARM et les IBM Power). Ce mode d'exécution, couplé à l'évolution technologique, a relancé la course à la puissance et n'a été freiné que par la chaleur dégagée. Grâce à un traducteur matériel en ligne, les codes complexes se sont trouvés être transcrits en codes spécifiques, adaptés au matériel, ce qui donne une quasi indépendance entre l'architecture interne de la machine et son code instruction qui devient donc une sorte de langage de programmation de bas niveau n'ayant plus qu'un rapport lointain avec la structure physique du processeur.

Avec ce regain en performance des CISC, les RISC perdirent leurs avantages, et beaucoup d'entre eux rejoignirent le musée des idées de l'informatique. Quelques-uns (ARM et POWER d'IBM) adoptèrent des instructions plus complexes qui leur permirent de devenir aussi efficaces que les CISC.

À titre de conclusion

L'informatique est une pure production humaine ; une myriade d'autres chemins auraient pu être pris pour traiter l'information. Comme d'autres productions humaines, elle est soumise à des effets de mode. Longtemps celle-ci a orienté ses recherches vers l'étude de processeurs qui se voulaient plus pratiques et surtout plus performants. De manière sous-jacente, une âpre lutte s'est développée entre la complexité et la simplicité, toutes deux vues comme le meilleur moyen d'atteindre l'efficacité. En fait, l'expérience a montré que celle-ci se situait souvent dans les choix intermédiaires.

L'évolution des langages et formalismes a structuré la pensée, donc la mode informatique, d'où un effet certain sur l'architecture visible des processeurs qui ont inclus ces mécanismes, ou au moins leurs soubassements, dans leurs architectures. L'histoire aurait pu continuer encore longtemps si deux facteurs n'étaient pas venus modifier son cours.

En premier lieu, la thésaurisation des logiciels est devenue une nécessité vue l'énorme volume de logiciels produits. Celle-ci n'est possible que dans un contexte de stabilité des formalismes d'écriture des programmes, ce qui a poussé fortement les processeurs à être compatibles de manière ascendante (c'est-à-dire capable d'exécuter des programmes écrits pour leurs prédécesseurs), ce qui a fini par bloquer toute évolution des machines virtuelles.

En second lieu, l'arrivée des systèmes distribués et à fort parallélisme a reporté l'attention des chercheurs vers les techniques de parallélisation et d'interconnexion des processeurs. Ceux-ci sont devenus de simples moyens technologiques, dont la nature n'a plus vraiment d'importance.

Les machines-langage ont disparu face à la puissance et au coût des machines standards fabriquées en beaucoup plus grand nombre. Les machines-système se sont réduites aux familles x86-amd64. Celles-ci occupent une forte position dans toutes les branches de l'informatique, face aux ARM qui ont presque conquis la totalité du large marché des applications embarquées et portables.

Pour résumer, la mode et les chercheurs se sont désintéressés des processeurs pour se consacrer aux systèmes parallèles en utilisant les processeurs disponibles sans trop se soucier de leurs spécificités. Pour des raisons commerciales, ces survivants se sont réduits à deux voies concurrentes :

- une famille de processeurs-système : les x86 de 32bits qui ont évolué en amd64 de 64 bits, principalement fabriqués par Intel et AMD ;
- deux familles de RISC complexifiés : la famille des ARM issus de l'informatique embarquée et celle des Power développée par IBM.

Tous ces processeurs utilisent des techniques d'exécution parallélisées qui semblent buter sur une limite difficilement dépassable de 2,5 instructions par cycle d'horloge, laissant la technologie devenir l'unique facteur d'évolution. Seule, la famille des amd64 véhicule encore le concept de machine-système qui s'éteindra si l'évolution s'oriente vers l'autre voie, et ceci dans l'indifférence la plus totale. Les machines-langage et les machines-système constitueront alors une sorte d'antiquité de l'informatique pleine de beaux monuments…

Bibliographie

Anceau F. (2000). « De von Neumann aux super-microprocesseurs ». *In* Y. Michaud (éd.), *Université de tous les savoirs.* Vol 5, Paris : Odile Jacob.

Anceau F., Baille G. & Schoellkopf J.-P. (1977). « Machine informatique électronique destinée à l'exécution parallèle d'un langage post-fixé ». *Brevet n° 2 325 985,* ANVAR.

Bellec J., Chain T., Humblot D., Joly C. & Lepicard G. (1989). « De Charlie au DPS7000… 15 ans de BULL DPS7 ». Édition semi-interne Bull S.A., « Méthodes et Informatique ».

Böhm C. & Jacopini G. (1966). « Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules ». *Communications of the ACM* 9 (5), pp. 366-371.

Chevance R.J. (1974). « A COBOL Machine ». *ACM Sigplan Notice,* 9/8, New York : ACM, pp. 139-144.

Étiemble D. (1991). *Architecture des processeurs RISC.* Paris : Armand Colin.

O'Connor M. & Tremblay M. (1997). « Picojava-1: The Java Virtual Machine in Hardware ». *IEEE Micro,* 17/2, pp. 45-47.

Organik E.I. (1972). *Multics Systems: An Examination of its structure.* Cambridge : MIT Press.

Organik E.I. (1973). *Computer System Organization*. Academic Press.

Palus M. (2006). « Étude et validation de l'architecture d'une machine Java de hautes performances ». Thèse de Doctorat en informatique, Université Paris 6.

Mounier-Kuhn P. (2010). *L'informatique en France de la seconde guerre mondiale au plan calcul, l'émergence d'une science.* Paris : PUPS.

Schoellkopf J.-P. (1977). « Machine PASC-HLL : Définition d'une architecture pipeline pour une unité centrale adaptée au langage PASCAL ». Thèse de 3e cycle en informatique, INPG, Grenoble.

Tools B.A. (1992). *ADA, The enchantress of numbers: A selection from the letters of Lord Byron's daughter and her description of the first computer.* Mill Valley, Ca. : Strawberry Press.

Émergence des systèmes d'exploitation comme discipline

Claude Kaiser
Cédric, Cnam.

Résumé

Nous présentons ici le contexte dans lequel se sont développés les systèmes d'exploitation des ordinateurs et comment une nouvelle discipline a pu émerger à part entière pendant les décennies 1960-1970. Sont abordés successivement l'évolution très rapide et multiforme de l'informatique, les nombreux projets dans l'industrie en intense collaboration avec les centres de recherche, la variété des systèmes d'exploitation développés, le fort engagement académique avec des revues et des thèses, la reconnaissance de cette discipline en 1971 suivie de la publication de livres d'enseignement. L'article présente aussi l'effort français dans ce domaine et l'introduction de son enseignement au Cnam en 1974.

Mots-clés : systèmes d'exploitation, science informatique, industrie, centres de recherche, recherche et développement.

Ce qui suit est la chronique d'années de gloire pour les systèmes d'exploitation, relatée avec la subjectivité d'un témoin proche[1]. Cette vision de l'émergence de la discipline des systèmes d'exploitation des ordinateurs durant les décennies 1960-1970 s'appuie sur mon expérience professionnelle et sur mes archives.

En 1960 un calculateur digital est volumineux, il remplit plusieurs armoires techniques rangées dans une grande salle de calcul. Il est lent et a peu de mémoire. Par exemple l'IBM 7090 a un cycle de

1 Les sources utilisées pour rédiger cet article ont été essentiellement mes archives personnelles : bibliographies citées dans des thèses d'État, actes de séminaires et colloques organisés à l'IRIA (Institut de Recherche en Informatique et Automatique, devenu en 1979, et jusqu'à ce jour, l'Institut National de Recherche en Informatique et Automatique - INRIA), livres rédigés en collaboration. Elles furent complétées par quelques recherches sur le Web. La liste des archives originales est donnée en annexe.

base de 2 microsecondes et 128 kilo-octets de mémoire centrale. Le système d'exploitation qui, à l'époque, est défini comme « *l'intermédiaire entre l'utilisateur d'un calculateur et le matériel* » ne peut pas être un gros programme, faute de place en mémoire.

En 1975 un calculateur – il a changé de nom et s'appelle maintenant un ordinateur – occupe toujours une salle de calcul, mais il a gagné en puissance, il est plus rapide, il a plus de mémoire et plus de périphériques. Par exemple le VAX 11/70 de DEC[2] a un cycle de 200 nanosecondes et une mémoire centrale de 4 mégaoctets. L'ordinateur est devenu très complexe et un système d'exploitation est indispensable. Celui-ci est maintenant un programme de très grande taille, parmi les plus gros et les plus complexes réalisés. Il remplit désormais essentiellement deux fonctions complémentaires : fournir aux utilisateurs simultanés une interface plus commode que celle de la machine physique pour que chacun puisse programmer ou exploiter ses applications ; gérer les ressources de cette machine pour les allouer aux multiples activités qui se les partagent (Krakowiak & Mossière 2013 ; Kaiser 2015a).

Selon le dictionnaire *Le Robert*, le terme « discipline » est utilisé depuis le XV^e^ siècle pour désigner une branche de la connaissance, une matière enseignée. Une discipline scientifique émerge à la suite de nombreuses réalisations et expérimentations techniques et scientifiques, dans un domaine bien délimité, suivies de réflexions pour dégager les abstractions, les lois ou les concepts communs présents dans ces travaux. Quand cet ensemble commun de connaissances a recueilli l'adhésion des principaux protagonistes, il peut faire l'objet d'un enseignement destiné à transmettre le savoir et à former de nouveaux acteurs.

Appliquées au domaine des systèmes d'exploitation, ces étapes vont guider ma présentation. Après avoir indiqué les premiers grands projets qui ont utilisé des calculateurs dès 1960, on montre le développement de 1956 à 1978 de systèmes d'exploitation de divers genres, tant dans l'industrie que dans les universités, le fort engagement académique qui l'a accompagné et la reconnaissance formelle de cette nouvelle discipline en 1971. On verra comment cette discipline est née aux États-Unis et que la participation des Européens a surtout été britannique. On se penchera néanmoins sur les travaux faits en France et l'introduction d'un enseignement au Cnam.

Pour cette chronique je me suis restreint à ce que je connais le mieux, écartant des aspects importants comme l'évolution de l'architecture matérielle des ordinateurs, les bouleversements du contexte commercial et la mutation des milieux professionnels et sociétaux.

2 L'article de Paloque-Berges & Petitgirard, ainsi que l'entretien avec Gérard Florin, un collaborateur proche de C. Kaiser au Cnam, reviennent sur l'importance des machines de DEC dans l'histoire de l'informatique et des systèmes (cf. le premier volume de ce double numéro).

Dès 1950, des projets industriels et gouvernementaux de très grande taille utilisent des calculateurs digitaux

Aux États-Unis, dès 1950, de grands projets militaires et civils font appel à ce qu'on appelle encore à l'époque des « calculateurs digitaux » (*digital computers*). Des financements privés ou publics via la DARPA (Defense Advanced Research Project Agency), la NSF (National Science Foundation), soutiennent les recherches nécessaires. Participent à ces projets des sociétés comme Univac, IBM, TRW, Burroughs, Control Data, RCA, General Electric. En Europe apparaissent plus tard des projets analogues. Tous ces projets, dont nous donnons quelques exemples dans l'encadré 1, s'appuient sur des calculateurs spécifiques et dans chaque projet des logiciels spéciaux sont développés pour piloter ces calculateurs et les matériels connectés. Il n'y a pas encore de système d'exploitation bien individualisé.

Un contexte industriel évoluant rapidement

Développements constants de nouveaux calculateurs

Pendant ces décennies 1960 et 1970, on assiste à des progrès technologiques rapides et importants (avec en 1958 les circuits intégrés, en 1963

Quelques projets américains	
1950	Système SAGE, surveillance aérienne USA, avec matériel IBM.
1950	Naval Tactical Data System (NTDS), avec machines Univac.
1960	Système SABRE, pour la réservation aérienne (matériel IBM).
1965	Electronic Switching System de Bell Telephone, par Western Electric.
Quelques projets européens	
1959	CERN, système de contrôle du PS (Proton Synchrotron).
1963	SENIT pour la marine française de surface, HALIOTIS pour les sous-marins.
1967	EUROCONTROL, contrôle aérien en Europe.
1968-73	SNC Système National de Conduite du dispatching EDF (avec CAE 9040).

Encadré 1 – Projets des années 1960–1970 avec des calculateurs digitaux

le CMOS[3]). Selon la « loi de Moore », le nombre de transistors dans une puce double tous les deux ans. Il en résulte une croissance exponentielle de la performance de l'unité centrale et de la taille des mémoires, une complexification de l'architecture des ordinateurs et un accroissement de leur fiabilité. Cela entraîne une extension explosive du champ de l'informatique, notamment de l'informatique de gestion, et une expansion continue du marché des calculateurs. La complexité croissante des ordinateurs et de leurs applications rend indispensable une aide pour leur exploitation.

Un flux régulier de systèmes d'exploitation nouveaux et variés dans l'industrie et la recherche à partir de 1956

Les industriels américains, en forte concurrence commerciale, ont construit des calculateurs dits universels, c'est-à-dire destinés à de nombreux contextes d'utilisation. Les systèmes d'exploitation, que les Anglo-Saxons ont appelés *operating systems*, doivent s'adapter à cette approche. En 1975 un système d'exploitation doit traiter trois aspects majeurs : l'accès à l'information, la gestion des ressources et la synchronisation des processus[4] concurrents. L'optimisation de l'architecture pour chacun de ces aspects requiert une structuration particulière des éléments du système d'exploitation. La cohabitation de trois hiérarchies peut entraîner des contradictions et il faut éviter que cela soit préjudiciable aux objectifs ou au fonctionnement du système. Selon qu'on donne plus d'importance à l'un des aspects, on aboutit à un type de système plutôt qu'à un autre. Ainsi à partir de 1956 sont développés des systèmes d'exploitation de conceptions diverses et de plus en plus complexes tant dans le domaine privé que public. Les sociétés commerciales comme IBM, Ferranti, Burroughs, DEC, CII, ou Bull demandent au système d'exploitation d'optimiser le flux des programmes à traiter séquentiellement (*batch processing*) tandis que des universitaires, au MIT, à Berkeley, à Michigan, expérimentent l'accès direct au calculateur par des utilisateurs simultanés (*time-sharing*).

Les premiers systèmes d'exploitation considèrent un seul programme à la fois puis de plus en plus de programmes sont traités simultanément, passant donc de la monoprogrammation à la multiprogrammation. Une très étroite coopération entre les concepteurs de systèmes commerciaux et les chercheurs universitaires existe, et cette particularité américaine engendre des résultats significatifs. L'encadré 2 donne un aperçu chronologique de l'apparition des systèmes d'exploitation avec quelques jalons importants et montre la grande variété des approches. On fait apparaître entre parenthèses les noms des principaux concepteurs connus. Cette modalité sera gardée pour la suite des présentations.

3 *Complementary metal oxide semi-conductor.*

4 Un programme est une entité passive, décrivant une suite d'instructions. Un processus est son pendant dynamique, une entité active qui représente l'exécution de cette suite d'instructions par l'ordinateur.

1956	le premier moniteur d'enchaînement : GM/NAA est réalisé pour IBM 704, (Patrick, Mock) suivi en 1959 par IBSYS pour IBM 7090/7094.
1961	le premier système en temps partagé, CTSS est réalisé au MIT sur IBM 7094 (Corbató, Dagget, Daley).
1961	Ferranti, à Manchester en Grande-Bretagne, introduit la pagination câblée pour l'ATLAS (Kilburn, Edwards, Lenigan, Sumner).
1962	le premier système multiprocesseur commercialisé l'est par Burroughs avec une machine et un système Algol, le MCP B5000/5500.
1964	la série 360 d'IBM est annoncée avec un ambitieux système OS/360, avec un moniteur d'enchaînement traitant en parallèle la sortie des résultats d'un programme terminé, l'exécution du programme en cours et l'entrée du programme suivant à exécuter (batch with spooling). Cette série va permettre à IBM de conquérir le marché, grâce à un excellent débit pour le traitement séquentiel des programmes et une politique commerciale efficace. Mais le projet, développé entre 1964 et 1967, prend un retard considérable qui doit être comblé par des systèmes moins ambitieux (comme DOS 360) et il a un coût faramineux (cf. Brooks, 1975). Il y aura plusieurs variantes (PCP, MFT, MVT).
1964	le projet GENIE de temps partagé est développé sur SDS 940 à l'université de Berkeley (Lampson, Thacker, Deutsch).
1965-1970	le système MULTICS est réalisé au MIT, avec des innovations majeures qui vont marquer la discipline (Corbató, Dennis, Denning, Saltzer, Schroeder, et de nombreux autres acteurs ; y collaborent les Français Pouzin et Bensoussan).
1967	le MTS, Michigan Time-sharing System, est construit pour un IBM 360-67 à l'université de Michigan (Arden, Galler, Pinkerton).
1967	IBM développe CP/CMS, générateur de machines virtuelles à Cambridge Massachussetts aux États-Unis et au centre scientifique IBM de Grenoble (y participent les Français Auroux, Bellino, Hans).
1969	aux Bell Labs est créé UNICS, mono-utilisateur présenté comme un clin d'œil à MULTICS multi-utilisateur (Ritchie, Thomson). Il sera rapidement suivi par l'UNIX multi-utilisateur.
1970	TOPS-10, puis TENEX (BBN) sont développés pour DEC PDP 10 (Bobrow, Murphy), qui, en dépit de leur faible diffusion commerciale, ont influencé les méthodes de conception et de développement des systèmes.
1970	en France Honeywell BULL annonce la série 60 avec le niveau 64, puis GCOS/64, GCOS7 transactionnel (Lepicard, Carré, Slosberg, Bellec).
1970	en France CII réalise SIRIS 7 pour le CII 10070, ex SDS Sigma 7, puis en 1972 SIRIS 8 pour l'IRIS 80 (réécriture du système par Ichbiah).
1972	IBM annonce VM/370.
1973	au centre de recherche Xerox PARC, l'ALTO OS (Thacker, MacCreight, Lampson, Soltis, Hoffmann, Metcalfe) révolutionne l'interface utilisateur.
1974	Digital Research commercialise CP/M un système d'exploitation pour Intel 8080, l'un des premiers microprocesseurs (Kildall).
1975	l'université Carnegie Mellon étudie le système HYDRA pour multiprocesseur (Wulf, Levin, Pierson).
1978	le système 38 d'IBM (Soltis, Hoffmann) est commercialisé.

Encadré 2 - Jalons importants de 1956 à 1978

Quelques systèmes d'exploitation particuliers pour les applications en temps réel

À côté des calculateurs et des systèmes universels, on a continué à développer des systèmes particuliers conçus pour commander les procédés industriels qui ont des contraintes spécifiques comme le respect des temps de réponse ou des échéances de calcul. Pour ce faire, ces systèmes privilégient la gestion des processus, se restreignent à une allocation statique des ressources et conservent en mémoire centrale les données et programmes de l'application. Ils sont de plus petite taille que les systèmes généraux et peuvent être embarqués dans le matériel de l'application. L'encadré 3 indique quelques réalisations. Certains systèmes comme le SIGMA 7 de XDS ou le VMS de DEC ont aussi comme objectif les applications scientifiques et sont plus proches des systèmes universels.

Développements de langages de haut niveau utilisables pour l'écriture de systèmes

Les premiers systèmes d'exploitation étaient simples et écrits en langage d'assemblage, représentation symbolique du langage machine binaire. Puis le système d'exploitation est devenu un programme de très grande taille et très complexe (il doit en particulier gérer l'exécution de programmes concurrents et leur synchronisation, aspect jusqu'alors inconnu et qui est l'objet de recherches). Écrire ce programme en langage d'assemblage est devenu un exercice très difficile et sujet à de nombreuses erreurs. Or durant cette période ont été dévelop-

1961	D-17 Autonetics (guidage Minuteman).
1967	RC 4000 de Regnecentralen au Danemark : multiprogrammation avec des processus communiquant par messages (Brinch Hansen).
1968	DEC PDP 8 mini-ordinateur.
1968	CTL (GB) Modular One calculateur à 16 bits, système multitâche, communication par messages, sémaphores… (écrit en assembleur).
1969	XDS 940 puis SIGMA7 pour le temps réel industriel et le calcul scientifique.
1970	Hewlett Packard HP3000.
1972	PLESSEY(GB) System 250 (*capabilities* - England).
1972	RTOS Data General.
1972	DEC RT-11 pour PDP11.
1976	RTES pour Solar de Télémécanique (Grenoble).
1977	DEC VAX VMS pour le temps réel industriel et le calcul scientifique.
1981	Wind River VxWorks (utilisé par la NASA dans le robot envoyé sur Mars).

Encadré 3 – Quelques systèmes conçus pour le temps réel

pés des langages informatiques dits de haut niveau, c'est-à-dire plus proches du langage naturel : Algol (1960), BCPL (1960), PL1 (1963), PL360 (1968), Pascal (1970), MESA (1970), C (1972). Les systèmes sont dès lors programmés en langages de haut niveau.

Dispositifs spéciaux pour les systèmes d'exploitation

Le développement de systèmes d'exploitation de plus en plus performants et fiables devient une nécessité incontournable et un facteur de vente. L'architecture des nouveaux calculateurs s'accompagne de dispositifs câblés spécifiques devenus nécessaires pour les systèmes d'exploitation. Les plus importants, ceux qui sont indispensables tant pour le partage de l'ordinateur que pour la protection entre des utilisateurs multiples, concurrents et concomitants, sont indiqués dans l'encadré 4.

Un riche bilan de systèmes de conception et d'architecture différentes

Parmi les systèmes d'exploitation de conception et de réalisations différentes, réalisés de 1956 à 1978 et cités dans les encadrés 2 et 3, certains ont eu une influence majeure pour la conception des systèmes ultérieurs et pour la définition de la discipline. Sont remar-

Horloge temps réel, programmable, différente du séquenceur de l'unité centrale (UC), pour mesurer le temps écoulé.

Mécanisme d'interruption (1956) et de commutation de contexte, complété par le déroutement, l'appel système, pour intervenir de l'extérieur sur le déroulement d'un programme.

Jeu d'instruction permettant la réentrance des programmes et le partage de leur code

Protection mémoire permettant de traiter séparément le code et les données.

Mode privilégié pour l'utilisation de l'UC par le Système d'exploitation (SE).

Canaux d'entrée-sortie autonomes avec accès direct à la mémoire (DMA).

Disques durs rapides (1956) pour le va-et-vient des programmes avec la mémoire centrale.

Pagination de la mémoire (1962), avec le traitement du défaut de page et la reprise possible et correcte d'une instruction dont l'exécution a été interrompue.

Aide à la gestion de la mémoire virtuelle, avec translation de mémoire ou segmentation, ou utilisation d'une mémoire associative.

Pagination et mémoire virtuelle permettent la cohabitation concomitante de nombreux programmes en mémoire centrale.

Encadré 4 - Dispositifs câblés inventés pour les systèmes d'exploitation

quables pour leur originalité le Burroughs MCP B5000/5500[5], première machine et premier système dont l'architecture est basée sur la sémantique d'un langage, ici Algol, le CP/CMS, première machine virtuelle par IBM, le RC 4000 de Regnecentralen avec une communication par messages entre processus, l'UNIX qui se distingue par la création de processus par copie du contexte du processus créateur, la communication par messages, et l'enchaînement (*pipes*) de processus ou encore l'ALTO qui révolutionne l'interface homme-machine. Le succès commercial de l'OS/360 avec MFT et MVT va permettre à IBM de gagner environ 90 % du marché de l'informatique de gestion et d'en être le leader incontesté tandis que VAX/VMS sera le modèle emblématique de Digital pour conquérir les marchés industriels et scientifiques. TENEX apparaît comme un modèle d'ingénierie et influencera beaucoup VAX/VMS. Enfin certains systèmes (CTSS, GENIE, MULTICS, CP/CMS) sont le résultat de projets de recherche qui comme on le verra dans le chapitre suivant sont accompagnés d'un fort engagement académique.

5 On lira avec profit les ouvrages d'E. I. Organick, sur Burroughs (1973) et sur MULTICS (1972).

Le contexte académique avec forte activité de recherche et développement scientifiques

C'est vers 1965 que la thématique des systèmes d'exploitation commence à être intégrée pour elle-même dans les programmes des centres de recherche tant publics que privés. Des journaux scientifiques, des colloques, des écoles de recherche sont consacrés aux systèmes d'exploitation. L'application d'une démarche scientifique fondée sur l'abstraction et la virtualisation, accompagnée par l'expérimentation de nouveaux concepts, permet de dégager des résultats généraux et un fondement commun de connaissances. La diffusion de ces résultats dans la communauté scientifique les rend accessibles à tous, permet la vérification de leur bien-fondé et leur confère une portée générale. Des thèses analysent les problèmes rencontrés et dégagent les aspects majeurs de cette discipline naissante.

Une voie originale : « Capability Based Computer Systems »

En parallèle avec les voies commerciales et académiques qui vont mener à la définition de la discipline, il faut citer une approche originale centrée sur la protection de l'accès aux composants du système d'exploitation. Le système est structuré en éléments coopérants dont l'accès par un autre élément n'est possible que par une clé (*capability*) qui figure dans de la mémoire protégée et qui

1967	Chicago : Magic Number Machine (Fabry).
1968	Berkeley : CAL-TSS System (Lampson, Sturgis).
1972	Plessey (GB) : System 250 (England).
1973	Cambridge (GB) : CAP Computer (Wilkes, Needham).
1974	Carnegie Mellon University : HYDRA (C.mmp ; Wulf).
1978	IBM : System 38 (puis AS 400).
1981	Intel : IAPX 432.

Encadré 5 – Les systèmes d'exploitation à *capabilities*

n'est modifiable qu'en mode spécial. À chaque acteur est associée une liste de clés (*capability list*) qui identifie tous les éléments, matériels ou logiciels, auxquels il peut accéder et pour quelle utilisation.

Regroupés sous le vocable *capability based systems*, apparaissent des notions comme l'adressage uniforme, les objets, les domaines d'exécution et le type d'accès. Les résultats de cette voie de recherche ont été exploités par quelques industriels, comme Plessey, IBM et Intel. Les systèmes réalisés selon cette voie sont présentés dans le livre de Henry Levy, *Capability Based Computer Systems* (1984) qui en donne une description très complète.

Création de centres de recherche industriels travaillant sur les systèmes

Étant donné l'importance fondamentale du développement de systèmes d'exploitation, des centres de recherche industriels ont été créés spécialement. Citons par exemple :

- 1960 Centres scientifiques IBM : Yorktown Heights, Cambridge Scientific Center (CSC)
- 1967 Centre scientifique IBM Grenoble (Duby, Hans, Bellino, Auroux)
- 1970 Centre scientifique CII Grenoble (Bolliet, Verjus, Chupin, Balter)
- 1970 PARC, centre Xerox à Palo Alto (Taylor, Lampson, Kay, Thacker)

Prototypes de systèmes en Europe et en France

Le développement aux États-Unis de la recherche universitaire sur les systèmes d'exploitation a suscité en Europe des réalisations de prototypes.

À de très rares exceptions près, la collaboration exemplaire entre industriels et chercheurs publics aux États-Unis n'a pas eu lieu en France, les industriels français préférant à l'occasion passer des

En Europe	
1968	Eindhoven (Pays-Bas) : THE (Dijkstra, Habermann).
1968	Edinburgh (Grande-Bretagne) : EMAS pour ICL System 4-75 (Whitfield, Shelness, Stephens, Wight).
1971	Manchester (Grande-Bretagne) : MU 5 multiprocesseur (Sumner, Woods, Kilburn, Morris, Rohl).
1973	Cambridge (Grande-Bretagne) : CAP Computer (Wilkes, Needham).
En France	
1966	Grenoble : Système en temps partagé 1401/7044 IBM (Auroux, Bellino).
1968	Grenoble : DIAMAG2, Système conversationnel à accès multiple (Bellot, Siret, Verjus).
1967	CERA SupAéro Paris : Système SAM sur CII 10070 (Rossiensky, Tixier).
1968-1972	IRIA Rocquencourt : Système ESOPE sur CII 10070 (Bétourné, Ferrié, Kaiser, Krakowiak, Mossière)[6].
1969	Paris : Un système de multiprogrammation sur NCR 4130. (Girault).
1972	Rennes : Système SAR (Verjus, Trilling, André, Herman…).
1972-1975	CII Louveciennes : Projet Y (Derville, Chevance, Mansion).
1975	Grenoble : Système GEMAU (Guiboud-Ribaud, Otrage, Briat, Balter, Rousset de Pina) sur IRIS 80.

Encadré 6 – Quelques prototypes de systèmes d'exploitation en Europe

contrats avec les universités américaines[7]. L'encadré 6 indique quelques prototypes de systèmes d'exploitation réalisés en Europe pendant cette période.

Publications scientifiques, congrès, écoles de recherche

À partir de 1967, les principales sociétés savantes en informatique ont créé des publications spécialisées ou donné de la place aux systèmes d'exploitation dans les revues déjà existantes.

Aux États-Unis cela concerne l'Association for Computing Machinery (ACM) avec les *Communications of the ACM* (*CACM*), *Journal of the ACM* (*JACM*), l'Institute of Electrical and Electronics Engineers (IEEE) avec

6 L'histoire de la réalisation de ce prototype a été publiée dans Bétourné & *al.*, 2004.

7 Parmi ces rares exceptions, en 1962 la collaboration de la société Mors, devenue ensuite Télémécanique, avec le laboratoire d'automatique de Grenoble et en 1967 et 1970 la collaboration des centres scientifiques IBM et Bull de Grenoble avec l'IMAG (Institut d'informatique et mathématiques appliquées de Grenoble).

les *Transactions on Computers*, puis les *Transactions on Operating Systems*, et IBM qui crée l'*IBM Systems Journal* en 1962. En Grande-Bretagne, la British Computer Society initie *The Computer Journal*.

En France, l'Association française de Calcul et de Traitement de l'Information (AFCALTI) devenue en 1964 l'Association Française d'Informatique et de Recherche opérationnelle (AFIRO), puis en 1970 l'Association Française pour la Cybernétique Économique et Technique (AFCET), voit sa revue *Chiffres* devenir en 1967 *Revue française d'Automatique Informatique et Recherche Opérationnelle* (*RAIRO*). Elle accueille dès 1967 des publications sur les systèmes d'exploitation.

Des colloques spécialisés, des sessions entières de congrès, des écoles de recherche sont désormais dévolus aux systèmes d'exploitation.

Thèses sur les systèmes d'exploitation

Les premières thèses de doctorat (PhD) sur les systèmes d'exploitation sont soutenues aux États-Unis à la suite des prototypes réalisés dans les universités (MIT, Berkeley et Michigan principalement). Quelques thèses parmi les

CACM en 1967 : Premier SOSP (Symposium on Operating Systems Principles).

Congrès IFIP avec forte présence d'articles sur les systèmes dès 1968.

Grenoble en 1966 : atelier sur « l'utilisation des ordinateurs en temps réel et temps partagé », organisé par Bolliet.

École d'été CEA-EDF-IRIA en 1969 sur les systèmes d'exploitation (avec Dijkstra, Randell et Whitfield) au Bréau-sans-Nappe (France) (cf. figure 1 ci-après).

École d'été CEA-EDF-IRIA en 1972 sur les modèles et mesures de systèmes (avec Coffman, Pinkerton et Wescott) au Bréau-sans-Nappe (cf. figure 2 ci-après).

IRIA Rocquencourt en 1974 : Premier colloque international sur les aspects théoriques et pratiques des systèmes d'exploitation (Arden, Gelenbe, Habermann, Kaiser, Krakowiak, Randell).

IRIA Rocquencourt en 1974 : Workshop on protection in Operating Systems (Ferrié, Kaiser, Lanciaux, Martin).

Congrès AFIRO, AFCET avec forte consonance systèmes dès 1967.

AFIRO, AFCET : séminaire Structure des machines, animé par Boucher dès 1964.

IRIA : séminaire Structure et programmation des calculateurs de 1971 à 1974.

Encadré 7 – Congrès, colloques, écoles, séminaires

Figure 1 - École d'été CEA-EDF-IRIA de 1969 sur les systèmes d'exploitation.
Conférenciers : E.W.D. Dijkstra, B Randell, H. Whitfield

Figure 2 - École d'été CEA-EDF-IRIA 1972 sur les modèles et mesures de systèmes.
Conférenciers : E. Coffman, T.B. Pinkerton, B. Wescott

plus significatives sont indiquées dans l'encadré 8. Elles analysent les principaux problèmes rencontrés et dégagent les concepts majeurs de la discipline. Leurs titres sont expressifs. Elles sont les premiers témoins académiques de l'émergence de la discipline et diffusent les avancées obtenues. Toute l'industrie américaine en profite pour des réalisations futures. Figurent aussi dans cet encadré 8 quelques thèses européennes dans la mouvance des travaux américains.

Les premières thèses d'État en France dans cette discipline portent plutôt sur la conception, la réalisation et l'analyse de systèmes d'exploitation. On y trouve la mise en application des principaux concepts désormais reconnus par la discipline. Elles sont présentées dans les universités de Rennes, Grenoble et Paris.

Thèses de PhD aux États-Unis	
1964 Saltzer	Traffic control in a multiplexed computer system, MIT.
1965 Scherr	An analysis of time-shared computer systems, MIT.
1967 Lampson	Scheduling and protection on interactive multi-processor systems, Berkeley.
1968 Fabry	List-structured addressing, University of Chicago.
1968 Pinkerton	Program behaviour and control in virtual storage systems, Michigan.
1968 Denning	Resource allocation in multiprocess computer systems, MIT.
1970 Alexander	Time-sharing supervisory programs, Michigan.
1971 Holt	On deadlock in computer systems, Cornell.
1972 Goldberg	Architectural principles for virtual computer systems, Harvard.
1972 Schroeder	Cooperation of mutually suspicious subsystems, MIT.
1974 Redell	Naming and protection in extensible operating systems, Berkeley.
1974 Sturgis	A post mortem for a time-sharing system, Berkeley.
Thèses de PhD en Europe	
1967 Habermann	Harmonious cooperation of abstract machines, THE, Pays-Bas.
1972 Khaja	The implementation of the name store and the associated replacement algorithm in the MU5 computer, Manchester, GB.
1973 Walker	The structure of a well-protected computer, Cambridge GB.

Encadré 8 – Principales thèses de PhD

1973 Verjus	Nature et composition des objets d'un système de programmation. Rennes.
1973 Bellino	Conception et réalisation d'un système à accès multiple. Grenoble.
1973 Hans	Mécanismes pour systèmes générateurs de machine virtuelle. Grenoble.
1973 Kaiser	Conception et réalisation de systèmes à accès multiple : gestion du parallélisme. Paris.
1973 Krakowiak	Conception et réalisation de systèmes à accès multiple : allocation de ressources. Paris.
1973 Gelenbe	Modèles de comportement de systèmes informatiques. Paris.
1974 Anceau	Étude des systèmes hiérarchisés de ressources. Grenoble.
1974 Lenfant	Comportement des programmes dans leur espace d'adresse. Rennes.
1975 Guiboud-Ribaud	Mécanismes d'adressage et de protection dans les systèmes informatiques. Application au noyau Gemau. Grenoble.
1975 Brandwajn	Équivalence et décomposition dans les modèles à files d'attente. Application à l'évaluation des performances des systèmes d'exploitation. Paris.
1975 Ferrié	Contrôle de l'accès aux objets dans les systèmes informatiques. Paris.
1977 Mossière	Méthode pour l'écriture des systèmes d'exploitation. Grenoble.
1977 Leroudier	Systèmes adaptatifs à mémoire virtuelle. Grenoble.
1977 Potier	Modèles à files d'attente et gestion des ressources dans les systèmes. Grenoble.

Encadré 9 – Premières thèses d'État en France sur les systèmes d'exploitation

Et déjà au Cnam, des mémoires d'ingénieur

Les premiers mémoires d'ingénieur Cnam sur les systèmes d'exploitation sont réalisés dans l'équipe de recherche du Centre scientifique IBM de Grenoble et soutenus au centre régional Cnam associé.

- 1972 Le Heig : Espace virtuel sous CP/CMS, Cnam Grenoble.
- 1974 Dupuy : Working set dispatcher, Cnam Grenoble.

Définition et création de la discipline en 1971

L'industrie américaine est consciente de l'importance des systèmes d'exploitation. Un bon système est une nécessité et un atout important pour la vente de calculateurs. Il faut donc former de bons ingénieurs pour la conception et la conduite de ces systèmes. Devenu une branche reconnue de l'informatique sur le plan industriel, le monde des systèmes d'exploitation devient une discipline dans les instances

universitaires à la suite du rapport de 1971 de l'Académie nationale américaine d'ingénierie et de la publication de nombreux ouvrages d'enseignement supérieur.

An Undergraduate Course on Operating Systems Principles

L'Académie nationale américaine d'ingénierie a, grâce à un financement de la NSF (National Science Foundation), chargé un comité d'un rapport proposant le programme d'un enseignement supérieur des systèmes d'exploitation. Ce comité, le COSINE Committee[8], était constitué par d'éminents scientifiques du domaine et son rapport de 37 pages peut être considéré comme le document fondateur de la discipline. Ce comité était composé de six membres :

- Peter J. Denning, Chairman, Princeton University
- Jack B. Dennis, Massachusetts lnstitute of Technology
- Butler Lampson, Xerox Research Laboratory, Palo Alto
- Nico Haberman, Carnegie-Mellon University
- Richard R. Muntz, University of California, Los Angeles
- Dennis Tsichritzis, University of Toronto

Ils ont commencé par définir de façon plus précise le rôle des systèmes d'exploitation avant de proposer un enseignement articulé en huit points reprenant les concepts clés dégagés par les travaux académiques et reconnus par tous comme fondamentaux. Ce rapport capital est mis en ligne sur internet et disponible gratuitement (COSINE, 1971). Il situe les rôles du système d'exploitation selon le plan suivant :

1. *The system defines an extended language.*
2. *The system defines an extended machine, e.g., a « virtual machine ».*
3. *The system creates an environment for efficient program execution.*
4. *The system is an information management system.*

Cela définit les fonctions principales du système d'exploitation : étendre le langage machine, fournir à l'utilisateur une machine virtuelle plus commode d'utilisation, créer un environnement d'exécution efficace, gérer l'information du système et des utilisateurs. Le rapport propose en outre un enseignement articulé en huit points :

1. *lntroduction*
2. *Procedures*
3. *Processes*
4. *Memory Management*
5. *Name Management*
6. *Protection*
7. *Resource Allocation*
8. *Pragmatic Aspects.*

8 Computer Science in Electrical Engineering Committee.

Avant le COSINE
1964 WEGNER. *Introduction to system programming,* Academic Press.
1968 WILKES. *Time-sharing computer systems,* Nat. Acad. of Engineering.
1970 ARSAC. *Systèmes de conduite des ordinateurs,* Dunod.
1970 WATSON. *Time-sharing design concepts,* Mc Graw-Hill.
Après la parution du COSINE
1972 HOARE, PERROT. *Operating systems techniques,* Academic Press.
1973 COFFMAN, DENNING. *Operating systems Theory,* Prentice Hall.
1973 BRINCH HANSEN. *Operating systems principles,* Prentice Hall.
1974 MADNICK, DONOVAN. *Operating systems,* Mc Graw Hill.
1974 TSICHRITZIS, BERNSTEIN. *Operating systems,* Academic Press.
1974 SHAW. *The logical design of operating systems,* Prentice Hall.
1975 CROCUS. (nom collectif de Bellino, Bétourné, Briat, Canet, Cleemann, Derniame, Ferrié, Kaiser, Krakowiak, Mossière, Verjus). *Systèmes d'exploitation des ordinateurs. Principes de conception*. Dunod[9].
1975 GRAHAM. *Principles of systems programming,* Wiley.
1975 FREEMAN. *Software systems principles.* A survey, SRA.
1975 LISTER. *Fundamentals of operating systems,* MacMillan Press.
1976 HABERMANN. *Introduction to operating system design,* SRA.

Encadré 10 – Premiers livres d'enseignement de la discipline

De nombreux livres d'enseignement

Quelques livres pionniers avaient décrit des réalisations de systèmes. Mais après la publication du rapport du COSINE Committee en 1971 vont paraître désormais des ouvrages présentant les principes des systèmes d'exploitation. Dans les titres figurent des termes tels que théorie, principes, conception logique, fondamentaux, qui dénotent clairement la volonté d'enseigner une nouvelle discipline et pas seulement un lot de techniques. Ces ouvrages ne se contentent pas d'exposer les principes mais chacun d'eux est accompagné d'exercices qui permettent des mises en pratique de ces principes.

9 L'histoire de l'écriture collective de cet ouvrage est relatée dans CROCUS, 1993.

1971 : Une discipline scientifique : principes, concepts, théorie, expérimentations

Dès 1971, dans le COSINE et les ouvrages d'enseignement qui vont suivre, émerge désormais la matière d'un enseignement de la nouvelle discipline. Les principaux chapitres sont consacrés aux points suivants :

- Définition précise des rôles d'un système d'exploitation des ordinateurs.

- Conception hiérarchique ascendante avec architecture en couches au-dessus d'un noyau basique.

- Abstraction et virtualisation du matériel, avec les correspondances entre concret et abstrait :
  - processeur ➔ processus
  - canaux d'entrées-sorties ➔ processus
  - mémoire physique ➔ mémoire virtuelle
  - disques ➔ fichiers, espace virtuel

- Gestion du parallélisme entre des processus s'exécutant en concomitance et devant se synchroniser entre eux : paradigmes de cette synchronisation (exclusion mutuelle, lecteurs-rédacteurs, producteurs-consommateurs, repas des philosophes) et leur programmation.

- Gestion de l'information : représentation et accès, contrôle de cet accès et du partage de l'information, liaison (passage d'un nom désignant un objet dans un contexte particulier à l'emplacement où l'objet est stocké en mémoire).

- Gestion et optimisation des ressources : ordonnancement des processeurs, partage de la mémoire (pagination à la demande, algorithmes de remplacement de pages en mémoire centrale, va-et-vient entre la mémoire centrale et le disque de pagination), localité du comportement des programmes (analyse de leur utilisation de la mémoire au cours de leur déroulement), allocation d'un espace de travail en mémoire centrale (*Working Set*), choix et réglage du taux de multiprogrammation, optimisation des accès aux disques et aux mémoires secondaires de stockage.

- Permanence et qualité du service : évitement de l'écroulement (à la suite d'une allocation trop optimiste des ressources communes), prévention de l'interblocage (deux processus s'empêchent mutuellement de progresser), protection, tolérance aux pannes, sécurité, performances, mesures, modèles.

- Nécessité d'une démarche de génie logiciel (*Software engineering*) pour obtenir un système fiable, évolutif, adaptable.

Reconnaissance de la discipline

- *Attribution du prix Turing à des pionniers des systèmes d'exploitation*

Le prix Turing, prix scientifique de haut niveau considéré comme l'équivalent du prix Nobel pour l'informatique, a récompensé certains des acteurs de l'émergence de la discipline des systèmes d'exploitation des ordinateurs.

- 1972 Dijkstra (Algol, science et art des langages de programmation)
- 1980 Hoare (définition et conception des langages de programmation)
- 1983 Thompson, Ritchie (théorie des systèmes génériques et UNIX)
- 1990 Corbató (temps partagé, CTSS et MULTICS)
- 1992 Lampson (systèmes d'exploitation, environnements personnels, sécurité)

- *Classification ACM des systèmes d'exploitation*

L'ACM (Association for Computing Machinery), principale association savante en informatique, a publié en 1964 une première classification pour la *Computer Science*. Dans cette première classification les systèmes d'exploitation n'apparaissent pas sous leur dénomination mais cachés dans une sous-rubrique de la programmation, à savoir *« 4. Programming ; 4.3 Supervisory Systems »*.

À la suite d'une révision commencée en 1968, la nouvelle classification de 1991 leur attribue désormais une rubrique entière, sous l'intitulé *Operating systems* :

D.4	***OPERATING SYSTEMS (C)***
D.4.1	*Process Management*
D.4.2	*Storage Management*
D.4.3	*File Systems Management*
D.4.4	*Communications Management*
D.4.5	*Reliability*
D.4.6	*Security and Protection*
D.4.7	*Organization and Design*
D.4.8	*Performance*
D.4.9	*Systems Programs and Utilities*

Quel cours système au Cnam en 1974

En 1974 le département de Mathématiques et Informatique du Cnam m'a demandé de créer des cours sur les systèmes d'exploitation pour les cycles B et C hors temps ouvrable, et pour l'Institut d'Informatique d'Entreprise (IIE). Je les ai enseignés en utilisant comme support le livre CROCUS en cours de publication (Crocus, 1975). Les cours portaient sur les concepts des systèmes d'exploitation et les techniques associées et étaient accompagnés de quelques exercices dirigés. Les premiers textes polycopiés de cours comprenaient les points suivants :

- Processus : paradigmes du parallélisme, sémaphores, interblocage.
- Gestion de l'information : nommage, liaison, représentation, mémoire vir-

tuelle, CLICS (version pédagogique du système MULTICS).
- Allocation des ressources : ordonnancement des processeurs, des disques, pagination à la demande, localité, espace de travail, écroulement, taux de multiprogrammation.
- Protection : *Capabilities* et contrôle d'accès aux objets.

Le laboratoire de calcul du Cnam possédait un calculateur Modular One de la société britannique CTL (Computer Technology Limited) avec un mot de 16 bits, un système d'exploitation multiprocessus (appelé E4) avec une communication entre tâches par message et une synchronisation par sémaphores. C'était un système dans l'esprit du RC 4000, écrit en langage d'assemblage, malheureusement sans documentation du code source (c'était une politique délibérée de CTL). Il n'a donc pas pu être utilisé pour les exercices dirigés[10].

Conclusion

On s'est intéressé à l'émergence des systèmes d'exploitation pendant les décennies 1960-1970, en examinant les développements réalisés essentiellement aux États-Unis chez les constructeurs de calculateurs et dans les universités avec, à côté de ce lieu majeur, un fort concours de la Grande-Bretagne et des Pays-Bas. Après avoir indiqué les premiers grands projets qui ont utilisé des calculateurs dès 1960, on a montré comment le développement exponentiel de calculateurs nouveaux et variés, de 1956 à 1978, a été accompagné par des systèmes d'exploitation de divers genres, tant dans l'industrie que dans les universités. Parmi ces réalisations, certaines relevaient de centres universitaires et ce fut le début d'un fort engagement académique avec des publications et des thèses contribuant au passage vers l'abstraction et les concepts. Quand, vers 1971, la discipline fut jugée assez bien établie, l'adhésion des industriels et des chercheurs s'est concrétisée dans le rapport du COSINE qui installa formellement la discipline et son enseignement. Quelques données françaises, issues principalement de mes archives personnelles, permirent de montrer ce qui se passait sur un mode mineur en France. Enfin, le lecteur aura pu noter l'apparition d'aspects précurseurs, tels les systèmes virtuels, l'interface utilisateur ou les microprocesseurs, qui tous, au-delà de la période décrite, vont participer à l'effervescence continue de la discipline. Celle-ci n'est pas figée ![11]

10 Pour la recherche en systèmes au Cnam à cette époque, on verra l'article de Paloque-Berges & Petitgirard dans le premier volume de ce double numéro.

11 Remerciements : Je tiens à remercier les relecteurs qui m'ont aidé à rendre lisible ce qui était au départ une présentation orale s'appuyant sur des projections. Le lecteur trouvera bon nombre de ces explications sur les concepts de la discipline dans (Krakowiak & Mossière 2013, Kaiser 2015a) ou dans le glossaire ci-après.

Sources personnelles

Thèses d'État sur les systèmes d'exploitation

1967 Arie Nicolaas Habermann : « On the harmonious co-operation of abstract machines ». PhD Thesis. Promotor E.W. Dijkstra. Technische Hogeschool Eindhoven. Pays-Bas. 115 pages.

1973 Jean-Pierre Verjus : « Nature et composition des objets d'un système de programmation ». Thèse de doctorat d'État. Jury : M. Métivier (président), L. Bolliet, C. Coatmelec, C. Pair. Université de Rennes. 124 pages.

1973 Jacques Bellino : « Mécanismes de base dans les systèmes superviseurs : Conception et réalisation d'un système à accès multiple ». Thèse de Docteur es-Sciences appliquées Jury : N. Gastinel (président), L. Bolliet, J.-C. Boussard, M. Griffiths, S. Krakowiak. Université de Grenoble (remerciements de l'auteur à Max Peltier, directeur du Centre Scientifique IBM de Grenoble). 156 pages.

1973 Claude Hans : « Contribution à l'architecture de mécanismes élémentaires pour certains systèmes générateurs de machine virtuelle », Thèse de Docteur es-Sciences Jury : J. Kuntzmann (président), N. Gastinel, L. Bolliet, M. Griffiths, L. Nolin. Université de Grenoble (remerciements de l'auteur à Max Peltier, directeur du Centre Scientifique IBM de Grenoble). 123 pages.

1973 Claude Kaiser : « Conception et réalisation de systèmes à accès multiple : gestion du parallélisme ». Thèse de doctorat d'État. Jury : J. Arsac (président), C. Pair, P. Feautrier, H. Boucher. Université de Paris VI. 251 pages.

1973 Sacha Krakowiak : « Conception et réalisation de systèmes à accès multiple : allocation de ressources ». Thèse de doctorat d'État. Jury : J. Arsac (président), C. Pair, P. Feautrier, H. Boucher. Université de Paris VI. 213 pages.

1974 François Anceau : « Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques ». Thèse de Docteur es-Sciences Jury : J. Kuntzmann (président), J. Arsac, L. Bolliet, S. Krakowiak, G. Saucier. Université de Grenoble. 315 pages.

1975 Serge Guiboud-Ribaud : « Mécanismes d'adressage et de protection dans les systèmes informatiques. Application au noyau Gemau ». Thèse de doctorat d'État. Jury : N. Gastinel (président), L. Bolliet, C. Kaiser, S. Krakowiak, F.H. Raymond, J.-P.. Verjus. Université de Grenoble. 322 pages.

1975 Alexandre Brandwajn : « Équivalence et décomposition dans les modèles à files d'attente et leur application à l'évaluation des performances des systèmes d'exploitation ». Thèse de doctorat d'État. Jury : J. Arsac (président), C. Bétourné, S. Krakowiak, E. Mourier, M. Nivat. Université de Paris VI. 266 pages.

1975 Jean Ferrié : « Contrôle de l'accès aux objets dans les systèmes informatiques ». Thèse de doctorat d'État. Jury : J. Arsac (président), J.-C. Boussard, P. Feautrier, C. Kaiser, J.-P.. Verjus. Université de Paris VI. 134 pages.

1977 Jacques Leroudier : « Systèmes adaptatifs à mémoire virtuelle ». Thèse de doctorat d'État. Jury : N. Gastinel (président), C. Bocksenbaum, J.-C. Boussard, P. Denning, S. Krakowiak, J.-P. Verjus. Université de Grenoble. 403 pages.

Ouvrages concernant les systèmes d'exploitation

1971 Séminaires IRIA. « Structure et programmation des calculateurs ». Éditeurs S. Krakowiak et C. Kaiser. 217 pages.

1972 Séminaires IRIA. « Structure et programmation des calculateurs ». Éditeur C. Kaiser. 275 pages.

Bibliographie

Bétourné C., Ferrié J., Kaiser C., Krakowiak S. & Mossière J. (2004). « ESOPE : une étape de la recherche française en systèmes d'exploitation (1968-72) ». 7e colloque sur l'Histoire de l'Informatique et des Télécommunications. Cesson-Rennes novembre 2004, France. Rennes : Éditions Irisa/Inria-Rennes, pp. 173-198.

COSINE (1971). *An Undergraduate Course on Operating Systems Principles.* Task Force on Operating Systems (VIII) of the National Academy of Engineerlng (US) Report of the COSINE Committee of the Commission on Education.

CROCUS (1993). « Crocus : une étape dans l'enseignement des systèmes d'exploitation ». Colloque sur l'histoire de l'informatique. Sophia Antipolis, octobre 1993. AFCET.

CROCUS (collectif de Bellino, Bétourné, Briat, Canet, Cleemann, Derniame, Ferrié, Kaiser, Krakowiak, Mossière, Verjus) (1975). *Systèmes d'exploitation des ordinateurs. Principes de conception.* Paris : Dunod[12].

Brooks F. (1975). *The Mythical Man-Month*, Addison Wesley.

Levy H. (1984). *Capability Based Computer Systems.* Digital Press.

Organick, E. I. (1973). « Computer System Organisation : The B5700/B6700 series », Academic Press.

Organick, E. I. (1972). « The Multics System: an examination of its structure ». Cambridge, Mass. : MIT Press.

1973 Séminaires IRIA. « Structure et programmation des calculateurs ». Éditeur C. Kaiser. 328 pages.

1974 Séminaires IRIA. « Structure et programmation des calculateurs ». Éditeur C. Kaiser. 101 pages.

1974 « Operating Systems. Proceedings of an International Symposium held at Rocquencourt ». *Lecture Notes in Computer Science* 16, Springer Verlag. Edited by E. Gelenbe and C. Kaiser. 310 pages.

1974 « Protection in Operating Systems ». International Workshop. IRIA/LABORIA. Colloques IRIA. Edited by J. Ferrié, C. Kaiser, D. Lanciaux, B. Martin. 250 pages.

1978 J.-S.. Banino, J. Ferrié, C. Kaiser, D. Lanciaux. *Contrôle de l'accès aux objets partagés dans les systèmes informatiques.* Monographie d'informatique de l'AFCET. Éditions Hommes et Techniques, 104 pages.

12 Ouvrage disponible gratuitement sous format pdf sur cnam.cnum.fr (grâce à Pierre Cubaud) [URL : http://cnum.cnam.fr/CGI/redir.cgi?8CA2680].

Webographie

Krakowiak S. et Mossière J. (2013). « La naissance des systèmes d'exploitation. » [URL : https://interstices.info/naissance-systemes] consulté le 01/08/2017.

Krakowiak S. (2014a). « Les débuts d'une approche scientifique des systèmes d'exploitation. » [URL : https://interstices.info/jcms/int_70839/les-debuts-d-une-approche-scientifique-des-systemes-d-exploitation] consulté le 01/08/2017.

Krakowiak S. (2014b). « La naissance du génie logiciel. » [URL : https://interstices.info/jcms/ni_79198/la-naissance-du-genie-logiciel] consulté le 01/08/2017.

Krakowiak S. (2015) « Quelques aspects de l'histoire des systèmes d'exploitation », Séminaires Histoire de l'informatique et du numérique, au Cnam, conférence de janvier 2015, video [URL : http://www.musee-informatique-numerique.fr/] consulté le 01/08/2017.

Kaiser C. (2015a). « À quoi sert un système d'exploitation ? » [URL : https://interstices.info/jcms/p_83884/a-quoi-sert-un-systeme-d-exploitation] consulté le 01/08/2017.

Kaiser C. (2015b). « Le ballet des processus dans un système d'exploitation. » [URL : https://interstices.info/jcms/p_82112/le-ballet-des-processus-dans-un-systeme-d-exploitation] consulté le 01/08/2017.

Glossaire

Code : un programme comprend du code et des données (synonyme : instructions).

Désignation : utilisation de noms pour atteindre des objets du système.

Écroulement : effondrement du système suite à de mauvaises allocations de ressource.

Information : instruction ou donnée en mémoire.

Interblocage : situation où des processus s'empêchent mutuellement de progresser

Mémoire virtuelle : espace d'adressage d'un processus, virtualisation de la mémoire physique.

Multiprocesseur : calculateur avec plusieurs unités centrales ou processeurs.

Multiprogrammation : exécution concomitante de plusieurs processus.

Nommage : organisation des noms des objets en mémoire.

Noyau : partie centrale du SE, on dit parfois partie enfouie (en anglais kernel).

Objet : tout composant logiciel ou matériel, repéré par le système d'exploitation.

Pagination : découpage de la mémoire en pages de taille fixe.

Processeur : unité centrale, unité active, moteur du calculateur.

Processus : déroulement d'un programme séquentiel, abstraction du processeur.

Ressource : matériel ou logiciel, nécessaire pour dérouler un processus.

Sémaphore : mécanisme de synchronisation, introduit par Dijkstra.

Synchronisation : organisation du déroulement concomitant de processus parallèles.

Système d'exploitation (SE) : logiciel « chef d'orchestre » indispensable pour la bonne utilisation d'un ordinateur (*Operating System*).

Système de fichiers : ensemble ordonné de fichiers et de catalogues.

Temps réel : se dit lorsque l'exécution est contrainte par une échéance temporelle.

Temps partagé (*Time-sharing*) : accès simultané par plusieurs postes d'utilisateurs.

Virtualisation : démarche d'abstraction lors de la construction d'un système.

***Working set* ou Espace de travail :** espace minimal de mémoire centrale nécessaire pour la bonne exécution d'un processus à un instant donné.

The History of Unix in the History of Software

Thomas Haigh
University of Wisconsin - Milwaukee & Siegen University.

You might wonder what I am doing here, at an event on this history of Unix. As I have not researched or written about the history of Unix I had the same question myself. But I have looked at many other things around the history of software and this morning will be talking about how some of those topics, including the 1968 NATO Conference on Software Engineering, Algol, IBM SHARE and mathematical software in the 1970s, connect to the origins of Unix. As I worked to pull this presentation together I realized that some of those connections are clearer and more interesting than I had previously assumed, to the extent that they challenge us to reinterpret some of what we think we know about software history during this era[1].

The "software crisis" and the 1968 NATO Conference on Software Engineering

The topic of the "software crisis" has been written about a lot by professional historians, more than anything else in the entire history of software[2]. It is usually connected the to 1968 NATO Conference on Software Engineering, which is sometimes claimed to be this kind of very broadly based conference in which industrial managers, programmers, and academics came together. It is also sometimes suggested that this had enormous ramifications for our actual typical programming practice was conducted during the 1970s and later.

1 This text is the transcript of a keynote talk given at the International symposium "Unix in France and in the United States: innovation, diffusion and appropriation" organized at Conservatoire national des arts et métiers, Paris, France, on October 19th, 2017.

2 The software crisis and the NATO conference appear prominently in Mahoney (1988), Campbell-Kelly & Aspray (1996), MacKenzie, (2001) and several of the contributions to Hashagen & *al.* (2002).

I have a more specific sense of why those things were important. Suggesting that there is a connection of the Software engineering conference to the mainstream programming in corporations producing packages, accounting, is, according to me, an exaggeration. I believe that the perceived crisis at the end of the 1960s was very specifically in the development of systems software, primarily operating systems, which is a category, in that period, that is still tightly bonded with compilers and computer languages.

So what was the software crisis then? Looking back at the proceedings, we see that there were indeed, at the NATO conference, many people expressing concern about what was going on within the development of operating systems. Two systems in particular were of concern: IBM OS/360 (although none of the thousand people developing it were actually at the conference) and Multics. Several people developing Multics were at the conference, as well as the three partners of the project, MIT, General Electrics (GE), and Bell Labs, including Edward E. David who was one of the three managers with overall responsibility for the project. He represented Bell Labs and he wrote in his position paper for the conference:

> Among the many possible strategies for producing large software systems, only one has been widely used. It might be labeled "the human wave" approach, for typically hundreds of people become involved over a several years period… It is expensive, slow, inefficient, and the product is often larger in size and slower in execution than need be. The experience with this technique has led some people to opine that any software system that cannot be completed by some four or five people within a year can never be completed.

As Fred Brooks, who was overseeing the OS/360 development eventually, and famously, concluded: "*Adding manpower to a late software project makes it later*" (1975). To put his argument in economic terms, the marginal benefit that you get from having an extra-programmer is more than offset by the increase transaction costs in coordinating the work of a larger group of people. The high profile problems of such projects increased interest in finding better ways, which was to be called "software engineering", although nobody at conference knew what that might be.

I have a tangential argument that I would like to share, from my paper "Dijkstra's Crisis: The End of Algol and the Beginning of Software Engineering."[3] This was written back in 2010 as part of the *Software for Europe* project, for inclusion in a since abandoned book project. I noted that the specific phrase "software crisis" doesn't actually appear in any of the quoted dialog or position papers for the conference, though it does appear in the introduction to the proceedings. My sense is that it spread and took its modern association with Dijkstra's Turing award Lecture

3 See the draft version [URL: http://www.tomandmaria.com/Tom/Writing/DijkstrasCrisis_LeidenDRAFT.pdf].

"The humble programmer" (1972), a title which can be read as a rebuke to the pretentiousness of the idea of being a "software engineer". In a way, this cements retroactively the idea that the most important thing that happened at the conference was the declaration of a "software crisis" to which Dijkstra proposed some solutions.

You might wonder what Algol was doing in the title of my paper. The Algol effort launched in 1958, continuing through the 1960s, and was formalized in 1962 by IFIP Working Group 2.1. My view is that the Algol effort did more than any other project in the history of computing to create an international computer science research community. If you look at how the early Turing awards were given, then you see seven of them awarded to members of the original groups that defined Algol (in 1958 and 1960). But in the second half of the 1960s, things took a less happy turn. The group decided to go with a proposal (adopting a draft from Adriaan van Wijngaarden) for what became Algol 68, which was deeply controversial as it rejected a proposal from Wirth for what became Pascal. Tensions only get worse over the next few years and when in 1968 the new language (Algol 68) was approved, the group essentially fell apart. Eleven working group members either resigned or signed minority reports, expressing differences with the direction that the work took.

If you look at the chronology there, 1968 is the same year as the NATO Software Engineering conference. Thus my argument is that the NATO conference was to a large extent an attempt by those Algol "refugees" – who quit or wrote minority reports – in order to create a new community that would preserve that collaboration but take on a broader focus. Dijkstra wrote a minority report, signed by seven members. The interesting thing about that is that there were many specific criticisms circulating of Algol 68 as a language, but Dijkstra's dissent did not really address those at all. Instead it called for a philosophical kind of shift from traditional ideas about the purpose of a programming language towards the design of systems that would enforce good practice in the development of complex programs. There is a significant kind of overlap between the participants in the Algol efforts, particularly ones who were dissatisfied with Algol 68, and the group that were most active in the discussions of the 1968 conference and in editing the proceedings, which were enormously important in shaping how the conference was remembered[4].

In "Dijkstra's Crisis..." I argued that, looking at the trajectories of some of those individuals, including Dijsktra, Hoare, Randell, Perlis, Naur, Wirth, etc., an interesting kind of shared career trajectory emerges. Trained in science or mathematics, they worked in early 1960s developing systems software in small teams either in marginal manufacturers

4 13 of the 28 members of IFIP WG 2.1 active in design of Algol 68 attended the 1968 or 1969 conference. Naur and Randell edited the proceedings. Naur resigned from WG 2.1. Randell drafted a minority report opposing Algol 68 and signed Dijkstra's report.

(not IBM) or in research settings. They were very small groups, and they produced groundbreaking compilers and operating systems. By the time of the NATO conference, they were in a transition from writing the systems to working in corporate research labs or universities. So, among that community, the word had spread concerning the massive scale of IBM's OS/360 and TSS efforts and their lack of success – what Dijkstra called the "Chinese army" approach and its failure. I think it was instinctive for them to try to formalize and to institutionalize mathematical and logical approaches to the development of system software that came naturally from their training and their own experience as being successful in developing such systems some years earlier.

After the NATO 1968 conference and its much less successful follow up in Rome in 1969, my argument is that this core Algol dissident group essentially abandoned the "software engineering" project and instead found happiness in a different IFIP working group (2.3) on "Programming Methodologies", which was focused on theory. That is where ideas such as structured programming comes from. Randell wrote: "*we regarded this group, WG 2.3, as having twin roots, the Algol Committee and the NATO Software Engineering Conference*." (Randell, 2003)

Others of course picked up the term "software engineering", that they discarded, and repurposed it. Randell was critical of this and wrote:

> It was with little surprise to any of the participants in the Rome conference that no attempt was made to continue the NATO conference series, but the software engineering bandwagon began to roll as many people started to use the term to describe their work, to my mind often with very little justification. Reacting to this situation, I made a particular point for many years of refusing to use the term or to be associated with any event which used it[5].

I think that would capture the attitude of some Algol veterans as well. But, I now suspect that there is another kind of response that we can describe to the problems discussed at the 1968 NATO Conference, beyond the two I have just described (the people who picked up "software engineering" as an identity and the group that went instead in the direction of IFIP Working Group 2.3 and "formal methods").

I am going to illustrate this response by looking at Douglas McIlroy. His background had something in common with the other people I have described. Born in 1932, he had a PhD in applied mathematics, and spent his carrier (1958-1997) in Bell Labs, a research institution. He was head of the computing technology research group between 1965 and 1986. Bell Labs had been one of the core part of the development of Multics from 1964 onward. At the beginning of that period McIlroy was deeply involved with

5 "The 1968/69 NATO Software Engineering Reports" [URL: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/].

Multics, implementing the stopgap PL/1 compiler used in early Multics development work. The fact that the project had chosen a language for which no compiler yet existed gives you a sense of the challenges facing operating systems projects in the era. He was one of the more vocal participants in the 1968 NATO conference.

Looking at the chronology here, the year after that Bell Labs quit Multics, in April 1969, is the same year Unix developments begins. I am really interested in that early period of Unix history here. The first version of Unix (1971) was used internally for text processing. It was rewritten in C in 1973, and in 1975 it begins to spread widely outside Bell Labs to eventual world domination.

Unix as a Software Components Factory

McIlroy made a famous remark at the 1968 conference: "*We undoubtedly produce software by backward techniques. We undoubtedly get the short end of the stick in confrontations with hardware people because they are the industrialists and we are the crofters. Software production today appears in the scale of industrialization somewhere below the more backward construction industries*." Crofters are more or less Scottish peasants, so McIlroy is opposing very small scale, inefficient domestic production to a factory. His solution to this was to be something called "Mass Produced Software Components". Looking at the idea of the "software factory", an idea that was floating around at the conference, he argued that the idea of subassemblies was the most transferable part of an industrial production that corresponded with the idea of modularity[6]. He also said:

> My thesis is that the software industry is weakly founded, and that one aspect of this weakness is the absence of a software components subindustry…. The most important characteristic of a software components industry is that it will offer families of routines for any given job. No user of a particular member of a family should pay a penalty, in unwanted generality, for the fact that he is employing a standard model routine….

One of the best-known works on this history, at least among the historical community, is *The Computer Boys Take Over* by Nathan Ensmenger (2010). The dissertation it was based on was called "From "black art" to industrial discipline," a title that gives you a good idea of the arc of the story. In Ensmenger's telling, the software crisis and the 1968 conference were a crucial episode of the transformation of software from a "black art" practiced by eccentric and unmanageable virtuosos to a highly routinized kind of industrial production. He puts

6 "*Certain ideas from industrial technique I claim are relevant. The idea of subassemblies carries over directly and is well exploited. The idea of interchangeable parts corresponds roughly to our term 'modularity', and is fitfully respected.*"

McIlroy's paper at the conference at the center of that discussion, calling it "*blatantly management oriented*". According to Ensmenger "*McIlroy rejected the idea that large software projects were inherently unmanageable. The imposition of engineering management methods had enabled efficient manufacturing in myriad other industries, and would not fail to do the same for computer programming.*" (Ensmenger, 2010, p.197) He also write that McIlroy's "*vision of a software 'components factory' invokes familiar images of industrialization and proletarianization. According to his proposal, an elite corps of 'software engineers' would serve as the Frederick Taylors of the software industry, carefully orchestrating every action of a highly stratified programmer labor force.*" (*ibid.*)

So McIlroy has been cast by historians as the man who brought, or at least tried to bring, Taylorism into the heart for software production. Although I have been thinking for a while about the NATO conference for years, until last week I had not looked closely at what McIlroy actually wrote or at how he managed his own software projects. Now that I have done, I don't see any Taylorism in McIlroy's text. My reading is that his idea was to eliminate routine work with the aid of generalized routines, not to deskill programmers. His paper reads: "*The personnel of a pilot plant* [producing components] *should look like the personnel on many big software projects, with the masses of coders removed… There will be perhaps more research flavor included than might be on an ordinary software project, because the level of programming here will be more abstract.*" Among the development teams using these components to produce application systems, implicitly a much larger group, I believe that he thought that the computer itself takes over the routine parts of programming. During the discussion at the conference McIlroy quipped "*It would be immoral for programmers to automate everybody but themselves.*"

Admittedly the metaphors of components, factories, and industrialization were used rather incoherently at the conference (Mahoney, 2004). My sense is that, when he talked about industrialization by software components, McIlroy's idea was that it would allow the users of the components essentially to remain crofters, producing quick results in groups of no more than four or five people, which as McIlroy's former manager Edward E. David had noted was the only team configuration proven to be effective in software production during that era.

In sum, my thesis is that McIlroy is not talking about a Taylorized vision of deskilled programming labor, but… about something very much like the project he was about to manage: Unix. A world in which small teams of eccentric men with beards produced remarkable things, thanks to the benefits of reusable codes and modularity. Indeed, McIlroy's 1968 paper proposed Bell Labs as an organization that well equipped to produce software components:

> Bell Laboratories is not typical of computer users. As a research and development establishment, it must per force spend more of its time sharpening its tools, and less using them than does a production computing shop…. The market would consist of specialists in system building, who would be able to use tried parts for all the more commonplace parts of their systems.

McIlroy even identified text processing as an area where reusable tools were particularly important. He wrote: "*Nobody uses anybody else's general parsers or scanners today, partly because a routine general enough to fulfil any particular individual needs probably has so much generality as to be inefficient.*" And that of course was the original application of Unix a few years later.

When he oversaw the Unix project, McIlroy had the perfect opportunity to practice his beliefs on the management of programmer teams. In an oral history interview with Michael Mahoney in 1993, McIlroy said that "*one of the only places where I very nearly exerted managerial control over Unix was pushing for the inclusion of the pipe mechanism. […] From time to time," he would say 'How about making something like this?' and then put up another proposal.*"'. So firstly, this quote is hard to square with the idea that he ran projects like Frederick Taylor. By his own estimation he did little more than almost exert control on a few occasions, giving his talented team great autonomy. He was also a hands-on contributor to the project, writing some programs himself.

Secondly, the quote made me wonder about the connection of pipes, his main technical contribution to Unix, to his earlier musings about configurable software components. McIlroy seems to have been thinking about this for years. I found a document on the Internet, purportedly from October 11th 1964 in which McIlroy wrote: "*We should have ways of coupling programs like garden hoses - screw in another segment when it becomes necessary to manage data in another way*"[7]. This could be seen as an early statement of the idea realized with the pipe mechanism. In his oral history he said that from 1970 to 1972:

> one day I came up with a syntax for the shell that went along with the piping and Ken said, 'I'm gonna do it.' He was tired of hearing all this stuff…. He put pipes into Unix. He put this notation into the shell [Here McIlroy points to the board, where he had written f >g> c], all in one night…. [M]ost of the programs up until that time couldn't take standard input, because, there wasn't the real need. They all had file arguments…. Thompson saw that that wasn't going to fit into this scheme of things, and he went in and changed all those programs in the same night. I don't know how. And the next morning we had this orgy of 'one liners.' Everybody had another one liner. Look at this, look at that.

That's McIlroy's description of how pipes got into Unix. Mahoney asked: "*was there a notion of* […] *a Unix philosophy*

7 See [URL: http://doc.cat-v.org/unix/pipes/]. This is on an image of what appears to be a page excerpted from a larger document.

or a tool philosophy before pipes or did pipes create it?" According to McIlroy:

> Pipes created it…. The philosophy that everybody started putting forth: "*This is the Unix philosophy. Write programs that do one thing and do it well. Write programs to work together. Write programs that handle text streams because that is a universal interface.*" All of those ideas, which add up to the tool approach, might have been there in some unformed way prior to pipes. But, they really came in afterwards.

Mahoney had read McIlroy's 1968 paper and asked if this realized his hopes for software components. McIlroy said: "*if you if stand back, it's the same idea. But, it's at a very different level, a higher level than I had in mind.*"

As a historian I do, of course, tend to stand back. From that perspective I think the pipe mechanism was not the only thing that McIlroy contributed to the modularity of Unix. His choice not to impose heavy handed managerial control on the project encouraged or at least permitted the decentralized style that we associate with Unix and which comprises: loosely coupled parts, each part small enough to be understood easily by an individual; a relatively small kernel; different commands notoriously written by different people in different ways with different kinds of expectations for switches; the fact that you can choose your own shell, etc.

So McIlroy was a leading practitioner of the art of using technical rather than managerial mechanisms to coordinate the production of extremely complex software systems. One of the other things he is sometimes credited with is pushing the Unix team to develop the system of man pages[8]. Pipes allow interoperability without requiring common coding conventions or master planned structures. Likewise, man pages were a flexible and minimalist alternative to creating thousands of pages of documentation in advance of the creation of the system as the big projects of the 1960s have attempted.

Parallels between mathematical software and Unix

Another interesting thing was that McIlroy was clearly inspired by mathematical software. When he is describing the idea of reusable software components in his 1968 paper, his example, perhaps surprisingly, is a generator for sine routines.

8 For example, according to the Wikipedia page dedicated to the system of "man pages" [URL: https://en.wikipedia.org/wiki/Man_page]: "*The first actual man pages were written by Dennis Ritchie and Ken Thompson at the insistence of their manager Doug McIlroy in 1971.*" I do not vouch for the accuracy of this. McIlroy himself attributed the specific format of man pages to Dennis Richie, though that does not preclude the idea that he had prodded Ritchie to come up with something. He noted that "*During the system's first two years one literally had to work beside the originators to learn it. But the stock of necessary facts was growing and the gurus' capacity to handle apprentices was limited.*" (See [URL: http://www.cs.dartmouth.edu/~doug/reader.pdf].)

He also mentions mathematical routines as the area, as of 1968, where the reuse of components was most advanced. He mentions: "*Choice of Algorithm. In numerical routines, as exemplified by those in the CACM, this choice is quite well catered for already. For nonnumerical routines, however, this choice must usually be decided on the basis of folklore.*"

This must be put in the context I mentioned earlier, by going back to 1950s as I promised I would do, at least briefly, in order to look at mathematical software and user groups in the 1950s. McIlroy talked about several sources and possible producers of software components: "*What about the CACM collected algorithms? What about users' groups? What about software houses? And what about manufacturers' enormous software packages?*" He said that "*User's groups I think can be dismissed summarily and I will spare you a harangue on their deficiencies.*" That is all he had to say on the topic.

Why? Remember that McIlroy had been working with SHARE on the committee that was defining PL/1 back in 1964. SHARE was founded in 1956 as a cooperative group for "large" IBM scientific machine users. As its name suggests, it was intended to "share" programs, expertise, experiences and best practices, some similarities with the open source model as later defined. SHARE was composed of *ad hoc* collaboration groups for specific projects. It had mechanisms for code in its library, to share and respond to bug reports. It set up standards for coding and configuration of machines that would facilitate collaboration between different user organizations. It had open circulation of proposals and design documents. It also had the idea that there was a specific SHARE culture into which new IBM users had to be indoctrinated before they could be effective participants in the group. The software library had routines contributed by user sites and relied on IBM to catalog and reproduce the code. It classified the routine contributed according to a scheme that permitted to organize it. Contributors were supposed to be responsible for maintenance. On the standardization side you got a list of many standards that SHARE adopted, to share code and practices, to standardize machine configuration (setting of switches, control panels, etc.), to standardize system software (assembler and utility programs- not supplied by IBM), that leads to big project to create the "SHARE Operating System". As a well financed group they could afford to maintain the SSD – Share Secretarial Distribution, a mechanism for communication between meetings, mailing of large bundles of assorted materials (Committee reports, drafts for comments, letters, inquiries and responses, including bug reports; also microfilms of source code for programs). It was like the Internet RFCs or a Usenet Newsgroup, but in the 1950s and on paper.

The installation representatives were senior figures, responsible for multimillion dollars computing installations. Representatives often had an enginee-

ring or science background, and advanced degrees. They were responsible for the design and specification of software tools. SHARE representatives were able to commit employees to develop code, driven by the economies of scale in developing generic routines. One needed to accumulate a lot of library code and tools to make a computer usable and there was no proprietary advantage in developing something like a cosine routine.

When talking about Unix and the closely intertwined history of free software mechanisms for coordinating and distributing software of this kind, it is interesting to mention that SHARE had something very like the open source model in the 1950s. But IBM, Bell Labs, and other software producers actually backed away from it. Historian Atsushi Akera, discussing the problems with the "SHARE Operating System" project, after which the design of system software migrated to IBM (rather than being done within SHARE itself), called this "The Limits of Voluntarism" (Akera, 2001). But in mathematical software the transition for development effort from users to IBM was much less pronounced than in operating systems. Here SHARE remained a primary distribution mechanism until early 1970s.

In the 1970s, three models for packaging and distributing mathematics routines emerged. As McIlroy mentioned, this was supported by peer review and publications in journals, including CACM, as well as the commercial sale of software libraries. In the early 1970s, two companies, IMSL and NAG, began to sell numerical libraries. The creation of specialized packages by small teams of experts, e.g. LINPACK and EISPACK projects was also key to this emergence. Those are not actually exclusive choices. The same code was sometimes be made available in all three channels: published, distributed in non-commercial PACKS and incorporated with other routines into commercial numerical libraries.

From a history of science perspective, it is interesting to see arguments that the best way to improve the quality of the code was by peer reviewing it. This was tried in SHARE in the 1960s (the Numerical Analysis Project) although it ran into problems there. Peer reviewing a mathematical routine took a lot of time and expertise, so as a way of improving the SHARE library didn't work as well as people hoped. But peer review of software developed from there, thanks to a publication named *ACM TOMS* (*Transactions on Mathematical Software*) started 1975 by John Rice. This was created specifically as a publication venue for peer reviewed mathematical software (program source code was distributed via microfiche, card and tape). This was an important step because academic culture gave little credit for writing programs, still less for doing the work needed to produce high quality portable and robust code. As least code distributed in TOMS could be considered a publication (Haigh, 2010).

The PACK model is interesting to look at, beginning with EISPACK re-

leased in 1972, a package that computes eigenvalues and eigenvectors of matrices, giving standard routines in this area for a decade. Initially it was a FORTRAN conversion of Algol routines by James H. Wilkinson and Christian Reinsch (who had collaboratively produced them a few years earlier), which in turn implemented new, dramatically improved methods. The government grant for EISPACK was given explicitly to test a new methodology in software production. There is an interesting parallel to make with the approaches used on the Unix project to produce and distribute software. The PACK model was widely adopted[9], particularly with LINPACK, a sequel to EISPACK, which had a huge impact in super computing.

As with Unix, modularization of platform-specific code, to give high performance and portability, was a crucial part of this approach.

What happened there can be compared to the Unix phenomenon: experts producing the best routines they can on a particular purpose, which can then be black boxed and distributed, and used by other people. When I talked to the people involved with those projects, they didn't have the romanticized view of user driven involvement pushed by the likes of Eric Raymond. They didn't expect the users to be contributing code or particular insights. They wanted them to test the routines. They didn't expect them to fixing bugs in the code.

The other aspect of mathematical software that should be explored more fully as a parallel with Unix is the mixture of academic and commercial involvement, in a way that is different from the way we think of open source. The people involved had pragmatic interest in getting their code used by as many people as possible. Their salary was already paid by lab or university, they were not concerned with copyright or licensing arrangements. That's a parallel between the conditions at somewhere like Argonne National Laboratory and the specific conditions at Bell Labs in the 1970s. In many ways, it is an extension of social norms and practices of science where, according to which people are expected to share their work and data freely.

Conclusion

We can frame the history of Unix in different ways. From my viewpoint, the history of Unix is part of the history of operating systems, which is part of the history of software, which is part of the history of technology, which is part of… (insert additional levels here) … the history of the universe. You can approach in different ways as an example of different things.

We can't look at Unix in isolation. Specifically, here I have tried to sketch the mechanisms used in Unix, for the pro-

9 Dozens of specialized packages were produced within the labs during this era: FUNPACK, MUDPACK, FISHPACK, etc. (see Cowell, 1984).

duction and distribution of Unix that were inspired by, and have interesting parallels with those used in mathematical software from the 1950s onto the 1970s. The historical range of development and distribution practices can't be reduced to a simple "open vs. closed source" dichotomy.

I also think we should recognize that these quasi-academic environments like Bell Labs and the US National Labs are crucial for software development during this period. They are less focused on publication than universities; but on the other hand, they are less focused on products than regular corporate development groups.

In the era of the "software crisis", in the late 1960s, it was acknowledged that operating systems and compilers were very complicated. They were becoming too complicated for one or two people to code. Even earlier in the 1960s it was very hard for one or two people to take on a full-size operating system and compiler. This is why software had a bad record at that time: the few who could do systems software well tended to become famous computer scientists. However, by 1968 even the geniuses couldn't handle all the complexity involved in developing systems with all the functionalities which were expected for operating systems that could deal with time-sharing and the requirements of commercial software. In the 1968 conference proceedings, the need for technical architecture to modularize the production of system software appears as an obvious solution to this issue, as opposed to the "Chinese army" approach Dijkstra described. But figuring out a workable modularization method is hard.

We are left with an open question: is Unix the first solution to the "software crisis" that actually worked? It had the technological and social mechanisms to modularize the operating system; it created chunks that one or two very smart people could design and code. We can see its success in various areas as in producing generalized tools that embed the tacit knowledge to do the hardest stuff, like "lex" and "yacc". In the mid-1960s developing a good compiler was a huge challenge, and development delays and disasters were much discussed in the computer industry trade press and at the 1968 NATO conference. Thanks to these UNIX tools, and the new body of theory they embodied, by the end of the 1970s the creation of a compiler could be assigned as a class exercise to undergraduate students. Many attempts are later made to solve the same problem of creating the tools that people could use to produce complex software in modularized ways: design patterns, object oriented languages, application frameworks, source code repositories and version control systems.

Thus, we can understand Unix as a response to the broader software crisis, in the broader range of problems in the 1960s, and not just as an answer to the specific problems of Multics, as it is usually understood. We can also see the connection of the Unix project to the work

that was going on at Bell Labs around formal methods and compilers, would be very interesting for historians to dig into. The same goes for other ideas circulating at around the same time such as structured programming (the most famous IFIP WG 2.3 contribution), chief programmer teams, etc.

In that sense, Unix is an idea about the structuring and management of software creation. An idea that is realized by features embedded in the code itself, including pipes. We can also see Unix as a model for the distribution and packaging of software. The work of situating Unix within the broader history of software and computer science has barely begun.

Bibliography

Akera A. (2011). "Voluntarism and the fruits of collaboration: The IBM user group Share." *Technology and Culture*, 42(4), pp.710-736.

Brooks F. (1975). "The mythical man month." Proceedings of the international conference on Reliable software, Los Angeles, California – (April 21 - 23, 1975), ACM.

Campbell-Kelly M. & Aspray W. (1996). *Computer: A History of the Information Machine.* New York, NY: Basic Books.

Cowell W. ed (1984). *Sources and Development of Mathematical Software.* New York: Prentice-Hall.

Dijkstra E. W. (1972). "The humble programmer". *Communications of the ACM* 15.10, pp.859-866.

Ensmenger N. (2010). *The Computer Boys Take Over.* Cambridge (Mass.): MIT Press.

Haigh T. (2010). "John R. Rice: Mathematical Software Pioneer." *IEEE Annals of the History of Computing* 32, n° 4, pp.72-80.

Hashagen U., Keil-Slawik R. & Norberg A. L., eds. (2002). *Mapping the History of Computing: Software Issues.* New York: Springer-Verlag.

MacKenzie D. (2001). *Mechanizing Proof.* Cambridge, MA: MIT Press.

Mahoney P. S. (1988). "The History of Computing in the History of Technology". *Annals of the History of Computing* 10, no. 2, pp.113-125,

Mahoney P. S. (2004). "Finding a History for Software Engineering". *IEEE Annals of the History of Computing* 25, no. 1, pp.8-19.

Randell B. (2003). “Edsger Dijkstra”. *Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems* (WORDS’03F), IEEE Computer Society.

Unix : construire un environnement de développement de A à Z

Warren Toomey
The Unix Heritage Society (TUHS)
Traduction de l'anglais par Loïc Petitgirard[1].

Résumé

En avril 1969, alors qu'AT&T se retire du projet Multics, les chercheurs impliqués dans ce projet se voient privés de leur environnement de développement si « convivial ». Privés de leur « jouet », ces chercheurs ont commencé à prospecter pour un remplaçant. N'ayant rien trouvé d'approprié, Ken Thompson décide d'en écrire un, en partant de rien. Dès le milieu de l'année 1969 il crée un système d'exploitation autonome (self-hosting) sur un miniordinateur PDP-7 inutilisé. Il s'agit d'Unix, un système d'exploitation dont les systèmes actuels ont en partie hérité. Cet article s'intéresse à la création d'Unix, après qu'AT&T ait quitté le projet Multics, aux fonctionnalités et innovations de la version Unix créée sur le PDP-7, et au travail réalisé en 2016 pour remettre sur pied une version opérationnelle d'Unix PDP-7 sur la base du code source disponible.

Qu'est-ce qui a motivé le développement d'Unix ?

1969 n'a pas été une bonne année pour les chercheurs des Bell Labs de AT&T qui étaient impliqués dans le projet Multics. Multics était une tentative pour améliorer et développer un grand nombre de concepts du moment sur les systèmes d'exploitation (par exemple, la mémoire virtuelle, le multitâche) et de construire un système d'exploitation assurant un service continu comme pouvait l'être celui d'une entreprise électrique ou d'une compagnie téléphonique.

Mais Multics a souffert de l'effet de « second système » décrit par Brooks (1975) : il était trop technique et trop ambitieux pour atteindre tous ses objectifs sur la plateforme matérielle pour laquelle il avait été conçu, l'ordinateur General Electric GE-645. AT&T avait rejoint le projet Multics en 1964, avec General Electric et le MIT. En avril 1969, le projet

ayant dépassé les délais, AT&T décide de se retirer de Multics.

Cette décision laisse désœuvrés les chercheurs impliqués dans Multics (Ken Thompson, Dennis Ritchie, Doug McIlroy entre autres). Sandy Fraser décrit la situation aux Bell Labs dans un entretien avec Mike Mahoney[1] :

> Je rentrais dans le bâtiment et y trouvais une sorte de désarroi, un moral à coup sûr très bas. Il se trouve que j'étais le dernier arrivé. Il était clair que nous étions dans un moment de transition assez traumatisant pour bon nombre de gens… Le jouet [le GE-645] n'était plus là. La salle de l'ordinateur était vide. Les gens étaient tout simplement déprimés. Certains partaient. Il manquait clairement de l'entrain.

Pour s'être brûlé une première fois les ailes avec un projet de système d'exploitation, le management d'AT&T n'était pas prêt à soutenir de nouveaux travaux de recherches sur les systèmes d'exploitation. Mais les chercheurs avaient trouvé que le système Multics était un environnement de développement formidable et ils étaient motivés par l'idée de travailler sur une solution de remplacement adaptée à leurs besoins. Ritchie précise que[2] :

> Même si Multics ne pouvait pas accueillir beaucoup d'utilisateurs, il avait les qualités techniques pour notre projet, malgré des coûts exorbitants. Nous ne voulions pas perdre le petit créneau confortable que nous occupions, parce qu'il n'en existait pas d'autre équivalent : même les services de temps-partagé que les systèmes d'exploitation de GE allaient offrir plus tard n'existaient pas encore. Ce que nous voulions préserver, ce n'était pas seulement un bon environnement pour réaliser des programmes, mais un système autour duquel une communauté pouvait se former.

Les chercheurs du projet Multics aux Bell Labs ont tenté à de nombreuses reprises de persuader le management d'AT&T d'acheter une plateforme informatique plus modeste sur laquelle ils auraient écrit un système d'exploitation ; mais rétrospectivement, ces propositions exigeaient qu'AT&T dépense « *beaucoup trop d'argent pour trop peu de personnel avec un plan trop vague* » (Ritchie, 1980). En fin de compte, tous ces plans ont été rejetés.

Durant cette période sombre, Thompson était engagé dans deux activités qui allaient conduire au développement du système d'exploitation Unix. La première était le développement de Space Travel, une simulation informatique précise de la dynamique du Soleil, des planètes et des satellites du système solaire. Un joueur pouvait y naviguer dans un vaisseau spatial à travers le système solaire, voir le paysage et essayer d'atterrir sur les planètes et les satellites.

---

1 Entretien de M. Mahoney avec Sandy Fraser, 1989 [URL : https://www.princeton.edu/~hos/mike/transcripts/fraser.htm].

2 Entretien de M. Mahoney avec Dennis Ritchie, 1989 [URL : https://www.princeton.edu/~hos/mike/transcripts/ritchie.htm].

Space Travel a été initialement développé sur la plateforme GE-645, mais son fonctionnement était onéreux (75 $ de temps processeur pour un tour de jeu (Ritchie, 1980)) et son interface en ligne de commande laissait beaucoup à désirer. Après le retrait du projet Multics, Thompson a réécrit Space Travel pour le PDP-7, un ordinateur du laboratoire plus petit, qui avait été utilisé pour des travaux graphiques mais qui était devenu obsolète et excédait les besoins.

Ce travail a permis à Thompson de faire connaissance avec la plateforme PDP-7, qui était sévèrement contrainte à la fois en termes de mémoire (avec seulement 8,192 mots de 18-bits)[3] et d'espace disque (avec une capacité de stockage de seulement 1,024,000 mots), et une architecture du jeu d'instruction complexe. Ces restrictions ont en fin de compte influencé la conception du système Unix.

Le système de fichiers d'Unix

La seconde activité de Thompson après la fin du projet Multics était la recherche en conception et structuration des systèmes de fichiers. Thompson avait entamé cette recherche avant qu'AT&T ne se retire formellement de Multics.

> J'avais construit une simulation de haut niveau d'un système de fichiers complet. Je n'en étais pas au point où on donne des adresses, où la taille des fichiers peut être augmentée ou d'autres choses similaires. J'étais resté à un niveau supérieur. Je pense que nous n'en étions qu'à une ou deux séances de travail, Dennis [Ritchie], [Rudd] Canaday et moi. Nous ne discutions que d'idées générales sur la façon d'éviter que les fichiers ne s'emmêlent et les soucis liés à leur taille grandissante[4].

À partir de cette simulation de haut niveau d'un système de fichier, Thompson a pris le PDP-7 en main et a implémenté une structure du système de fichier. Au départ il s'agissait d'une simulation des fichiers et des actions sur ces fichiers, et au fur et à mesure le système simulé s'est étoffé pour devenir un système d'exploitation opérationnel.

> Pour faire fonctionner le système de fichiers, vous deviez créer, effacer et concaténer des fichiers, pour voir jusqu'à quel point il fonctionnait. Pour cela il fallait un scénario des commandes que vous deviez soumettre au système de fichiers, et le seul que nous avions était […] un ruban perforé qui disait : lit un fichier, écrit un fichier, ce genre de chose. Vous lanciez le scénario du ruban perforé et ça secouait un peu le disque. On ne savait pas exactement ce qui se produisait. On ne pouvait tout simplement pas le regarder, on ne pouvait pas le voir, on ne pouvait rien faire. Ensuite nous avons conçu quelques outils sur le système de fichiers. Nous avons

3 Un mot correspond à la taille de l'instruction standard utilisée par la machine.

4 Entretien de M. Mahoney avec Ken Thompson, 1989 [URL : https://www.princeton.edu/~hos/mike/transcripts/thompson.htm].

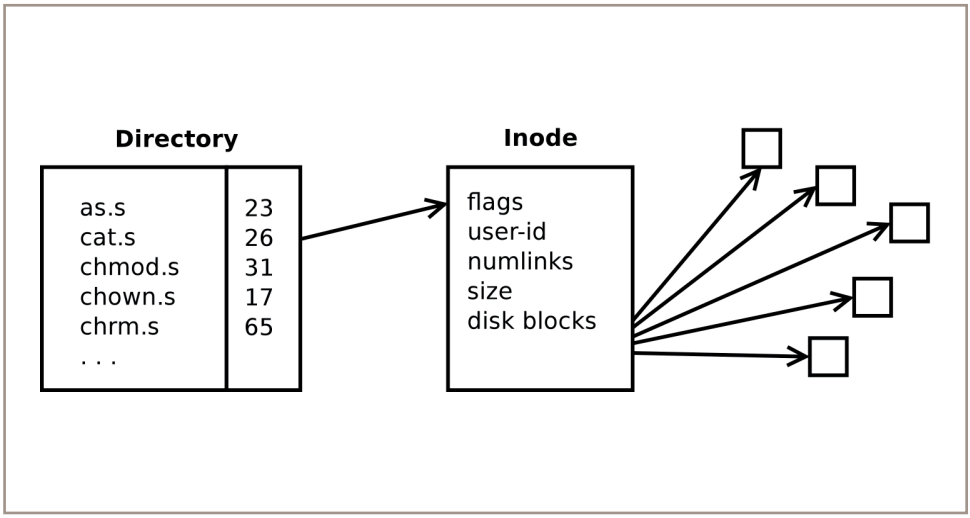

Figure 1 – Structure Inode sur Unix PDP-7

> utilisé ce ruban perforé pour charger le système de fichier et ses outils, et alors nous avons pu […] taper des commandes dans un outil qui s'appelait un interpréteur (*shell*), pour tordre le système de fichiers dans tous les sens, afin de pouvoir mesurer son fonctionnement et ses réactions. Le système de fichier primitif a duré peut-être un jour ou deux avant que nous ayons développé les outils dont nous avions besoin pour le charger[5].

Au XXIe siècle, nous voyons, à juste titre, les systèmes d'exploitation comme des systèmes extrêmement complexes, demandant des centaines de développeurs et des milliers d'heures pour être conçus. Au XXe siècle, Thompson ne le savait pas. Au lieu de cela, il raconte son été 1969 :

> Ma femme est partie en vacances en Californie pour voir mes parents. Elle est partie un mois en Californie et je me suis alloué une semaine pour chaque partie du système, le noyau, l'interpréteur, l'éditeur et l'assembleur, pour que le système puisse se reproduire lui-même. Pendant le mois où elle était absente, tout a été réécrit sous une forme qui ressemblait à un système d'exploitation, avec des outils de types connus : un assembleur, un éditeur et un interpréteur. Ça ne tenait pas encore la route, mais on y était presque[6].

En un mois de temps, Ken Thompson a pris un simulateur de système de fichiers et l'a réécrit pour en faire un système d'exploitation autonome (*self-hosting*) que nous connaissons aujourd'hui sous le nom de Unix PDP-7.

5 *ibid.*

6 *ibid.*

La structure du système de fichiers

Dès le départ, la structure du système de fichiers de l'Unix PDP-7 avait les marques distinctives qu'elle allait conserver le reste de sa vie (comme illustré sur la figure 1) :

- des *inode* (index) qui contiennent les métadonnées de chaque fichier ;
- des répertoires séparés de noms de fichiers, chacun pointant vers un *inode* ;
- un ensemble de blocs disque qui stockent les contenus de chaque fichier.

L'*inode* pouvait stocker jusqu'à sept numéros de blocs disque, pour des fichiers jusqu'à 7x64 = 448 mots de long. Pour des fichiers plus gros, le fichier était repéré dans le champ des « indicateurs » (*flags*) comme un grand fichier. Dans ce cas, les numéros de bloc pointaient vers des blocs disques qui chacun contenait 64 numéros de blocs disque, autorisant des fichiers jusqu'à 7x64x64 = 28 672 mots de longueur.

Le champ des indicateurs (*flags*) de l'*inode* identifiait aussi le type du fichier (fichier standard, répertoire ou fichier spécial), et les permissions pour les données (lecture et écriture pour le propriétaire du fichier, lecture et écriture pour tous les autres utilisateurs). Le propriétaire d'un fichier était identifié par le champ « *user-id* » ; le concept de « groupe d'utilisateurs » n'arrivera pas avant quelques années.

Le concept de lien physique a été une innovation du système de fichier de l'Unix PDP-7. Comme le nom d'un fichier était séparé des métadonnées de l'*inode*, cela permettait à deux ou plusieurs noms de fichiers de pointer vers la même métadonnée, comme l'illustre la figure 2.

Tous les noms de fichier reliés à l'*inode* étaient équivalents : aucun ne valait plus qu'un autre. Le champ « *numlink* » de l'*inode* enregistrait le nombre de noms de fichiers liés au fichier ; le fichier n'était supprimé du système de fichier que dans le cas où tous les noms de fichiers étaient déliés et que le compteur de lien tombait à zéro.

Les répertoires

Si l'organisation des contenus des fichiers et de leurs métadonnées était plutôt bien définie sur l'Unix PDP-7, on ne peut pas en dire autant de l'organisation des répertoires et de leurs relations entre eux.

> La première structure de système de fichiers ne dessinait pas une hiérarchie de répertoires mais un graphe orienté, et nous ne le limitions pas à un arbre. Nous expérimentions différentes topologies[7].

7 *ibid.*

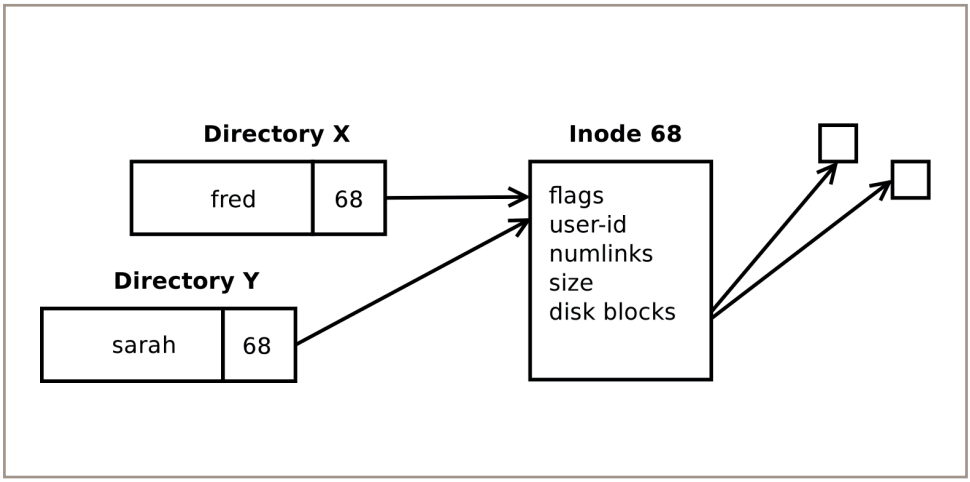

Figure 2 – Les liens physiques dans Unix PDP-7

Le noyau de l'Unix PDP-7 comprenait et gérait le concept de répertoire (contenant des noms de fichiers correspondants) et le fait qu'un répertoire puisse contenir des liens vers d'autres répertoires. Cependant, l'organisation des répertoires lui était indifférente.

Le premier système de fichiers Unix, et celui qui subsiste dans le système Unix PDP-7 restauré, contenait deux répertoires de haut niveau, « *dd* » et « *system* », comme l'illustre la figure 3. Le répertoire « *dd* » contenait les entrées du répertoire « *system* » et tous les répertoires « *home* » de l'utilisateur. Le répertoire « *system* » contenait tous les principaux fichiers exécutables du système et les fichiers spéciaux des périphériques. Aujourd'hui, « *dd* » correspondrait au répertoire « *root* », et « *system* » à une combinaison des répertoires « */bin* » et « */dev* ».

Pour naviguer dans le système de fichiers, chaque répertoire avait besoin d'un lien vers le répertoire « dd », pour y revenir. Et comme l'interpréteur était programmé pour chercher les codes binaires exécutables dans le répertoire « *system* », chaque répertoire devait contenir une entrée pointant dans le répertoire « *system* ».

Dans l'Unix PDP-7, le concept de chemin absolu comme */usr/local/bin/less* n'existait pas, ni celui de nom de chemin relatif (par exemple : *../../file ou sub-dir-1/subdir2/file*). Une fois que l'utilisateur était entré dans un répertoire, il ne pouvait faire référence qu'à des fichiers ou des répertoires visibles depuis ce répertoire. Ritchie souligne que :

> La commande « link » prenait la forme suivante :
>
> ln dir file newname

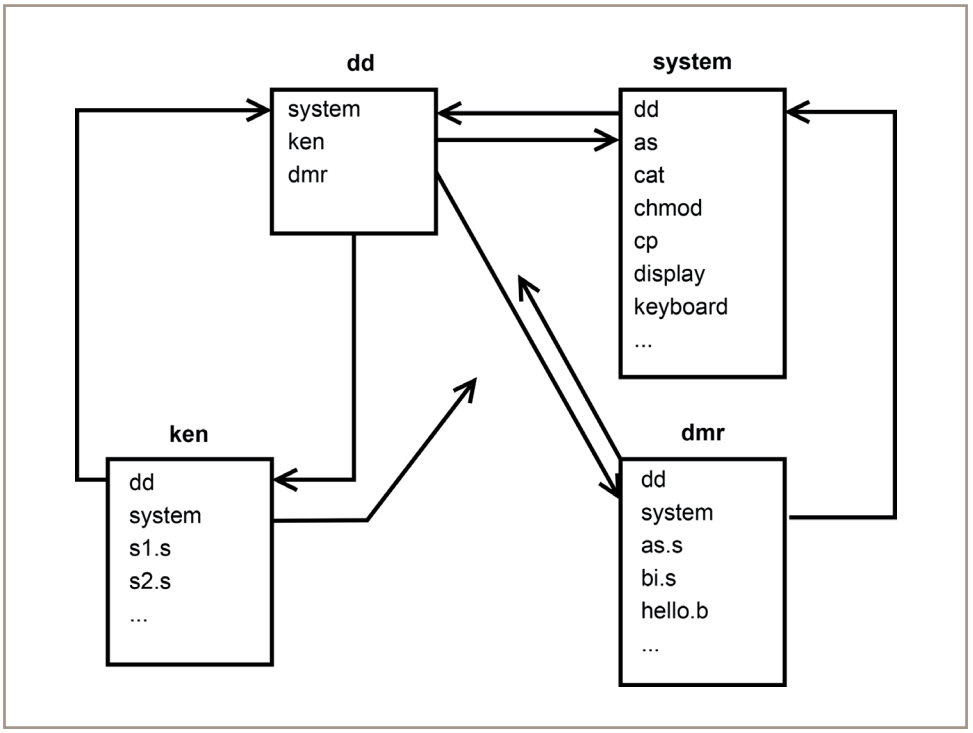

Figure 3 – Structure des répertoires sur Unix PDP-7

où « *dir* » était un fichier répertoire dans le répertoire en cours, « *file* » une entrée existant dans ce répertoire, et « *newname* » le nom du lien qui était ajouté au répertoire en cours. Parce que « *dir* » devait être dans le répertoire actuel, il est évident que les interdictions actuelles de créer des liens à des répertoires n'étaient pas imposées : le système de fichiers de l'Unix PDP-7 avait la forme d'un graphe général orienté.

Pour que chaque utilisateur n'ait pas à maintenir un lien vers tous les répertoires qui l'intéressaient, il existait un répertoire appelé « *dd* » qui contenait les entrées pour le répertoire de chaque utilisateur. Si j'étais dans mon répertoire « *home* » (« *dmr* »)[8], faire un lien vers un fichier x du répertoire « *ken* », je pouvais faire :

ln dd ken ken

ln ken x x

rm ken

Cela rendait les sous-répertoires suffisamment difficiles à utiliser pour qu'ils ne soient jamais utilisés en pratique. La convention « dd » rendait la commande « chdir » relativement commode[9]. Elle

8 NB : « *dmr* » correspond à Dennis MacAlistair Ritchie, et « *ken* » à Kenneth Thompson.

9 « chdir » pour « change directory ».

prenait plusieurs arguments et faisait passer du répertoire en cours au répertoire identifié. Ainsi :

chdir dd ken

faisait passer [du répertoire « *dmr* » au répertoire « *ken* » via le répertoire « *dd* »] (Ritchie, 1980).

Dans l'Unix PDP-11 (écrit en 1971), l'entrée « *dd* » dans chaque répertoire a évolué vers l'entrée « .. » (deux points) qui pointe vers le répertoire immédiatement supérieur. La mémoire supplémentaire disponible sur le PDP-11 permettait au noyau de gérer des chemins absolus et relatifs ; une fois que l'interpréteur pouvait trouver les exécutables dans le répertoire « */bin* », l'entrée « *system* » dans chaque répertoire n'était plus nécessaire.

Les opérations sur les fichiers et les fichiers spéciaux

L'Unix PDP-7 mettait en œuvre un ensemble d'opérations sur les fichiers immédiatement reconnaissables pour un programmeur Unix ou Linux actuel :

- Descripteur de fichier (*file descriptor*) = *open* (nom de fichier, mode)
- *close* (*file descriptor*)
- *read* (*file descriptor, buffer, amount*)
- *write* (*file descriptor, buffer, amount*)
- *seek* (*file descriptor, amount, whence*)

De cette manière, le noyau Unix s'abstrayait des détails du stockage d'un fichier et le remplaçait par un tableau linéaire de mots du PDP-7 (des octets dans les versions ultérieures d'Unix). L'Unix PDP-7 a étendu cette abstraction en introduisant le concept de « fichiers spéciaux ». Chaque fichier spécial représentait le contenu d'un périphérique auquel l'accès était possible en utilisant les opérations de fichiers ci-dessus, quelles que soient les caractéristiques du périphérique.

Le noyau d'Unix PDP-7 prenait en charge ces fichiers spéciaux, conservés dans le répertoire *system* grâce aux commandes :

- *ttyin* et *ttyout*, pour la console du PDP-7
- *keyboard* et *display*, pour l'écran Graphics-2 qui était utilisé comme un second terminal
- *pptin* et *pptout*, le dispositif de ruban perforé

Ainsi, sur un système de seulement 8 192 mots de mémoire, l'Unix PDP-7 était capable de fournir un environnement de développement multitâche pour deux utilisateurs.

Processus et contrôle de processus

L'Unix PDP-7 fournissait un environnement multitâche en divisant les 8K mots de mémoire en deux moitiés. La moitié basse de la mémoire était réservée au noyau. La moitié haute était mise de côté pour les processus en cours d'exécution. L'Unix PDP-7 basculait les pro-

cessus de la mémoire au disque durant le changement de contexte[10] entre deux processus en cours. Bien que cela permettait d'isoler les processus, le *hardware* du PDP-7 n'empêchait pas les processus d'accéder aux 4K de mots de la mémoire basse du noyau.

Ce que nous reconnaissons aujourd'hui comme l'ensemble canonique des mécanismes de contrôle de processus Unix (*fork()*, *exec()*, *exit()* et *wait()*) a évolué par étapes lorsque la version pour le PDP-7 a été développée puis réécrite pour la plateforme PDP-11.

Les commandes « *fork* » et « *exec* »

Thompson a emprunté et modifié le concept de *fork()* qu'il avait vu sur le système Project Genie sur l'ordinateur SDS930 lorsqu'il était étudiant en licence à l'Université de Berkeley. *Fork()* a été développé par Melvin Conway en 1962 comme une paire de primitives, *fork* et *join*, pour permettre la planification des processus d'un système multiprocesseur (Nyman & Laakso, 2016). Dans Unix PDP-7, *fork()* était utilisé par un processus pour créer une copie de lui-même qui pouvait ensuite être ordonnancée indépendamment.

Si *fork()* permet à un système de démarrer de nouveaux processus, ils sont cependant tous identiques. Une autre primitive était nécessaire pour permettre au système d'exécuter des programmes différents. Dans Unix aujourd'hui, c'est le mécanisme *exec()*, implémenté dans le noyau Unix, qui le réalise. Dans Unix PDP-7 ce mécanisme était implémenté dans l'interpréteur, comme illustré dans la figure 4.

Lorsque l'interpréteur du PDP-7 lisait une commande d'exécution d'un nouveau programme, il faisait d'abord une copie de lui-même avec un *fork()*. L'interpréteur originel attendait que le nouvel interpréteur s'exécute et se termine. Le nouvel interpréteur devait alors se remplacer par le nouveau programme demandé par l'utilisateur.

La fonction exécutée dans l'interpréteur se relogeait d'abord dans la partie supérieure de la mémoire, juste au-dessous des arguments du programme entrés par l'utilisateur dans la ligne de commande. La fonction relogée faisait alors un *open()* du fichier exécutable et un *read()* de son contenu, qu'elle transférait dans la mémoire utilisateur à partir de l'adresse 4096. Une fois l'exécutable chargé en mémoire, la fonction relogée faisait un *close()* du fichier et allait à l'adresse 4096 pour démarrer l'exécution du nouveau programme.

Ritchie fait remarquer que même ce simple mécanisme de contrôle de processus a évolué à partir d'une construction primitive :

10 Le « contexte » est l'ensemble des états des registres la machine, à un instant donné, qui permet l'exécution des processus.

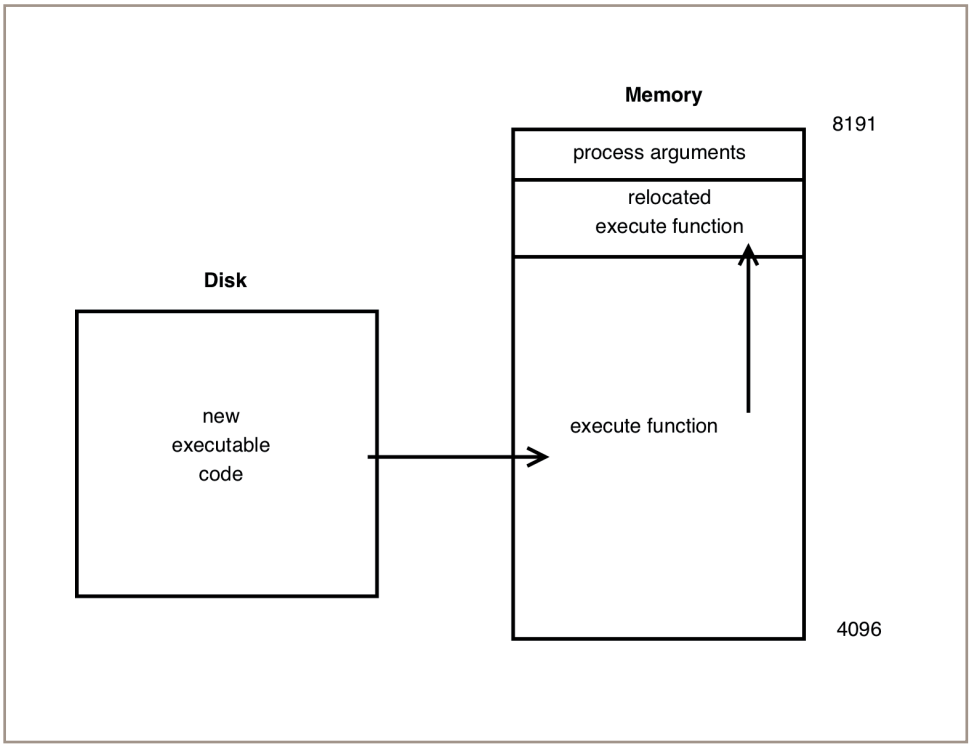

Figure 4 – Le mécanisme exec de l'interpréteur (*shell*)

Le contrôle de processus dans sa forme moderne a été conçu et implémenté en deux jours. C'est étonnant à quel point il s'adaptait au système existant ; en même temps, il est facile de voir comment les caractéristiques légèrement inhabituelles de la conception sont présentes précisément parce qu'elles représentaient des petits changements faciles à coder dans l'existant. La séparation entre les fonctions *fork* et *exec* en est un bon exemple. Le modèle le plus commun pour créer de nouveaux processus implique de spécifier un programme pour le processus à exécuter ; dans Unix, le processus créé par la commande *fork* continue de faire tourner le même programme que son parent jusqu'à ce qu'il appelle explicitement un *exec*.

La séparation des fonctions n'est certainement pas limitée à Unix, et était en fait présente dans le système de temps-partagé de Berkeley (Project Genie) qui était bien connu de Thompson. Néanmoins, il paraît raisonnable de penser qu'elle existe dans Unix principalement parce qu'il était facile d'implémenter *fork* sans changer grand-chose d'autre. Le système gérait à l'origine de multiples (c'est-à-dire deux) processus ; il y avait une table des processus, et les processus étaient échangés entre la mémoire principale et le disque. L'implémentation initiale de *swap* ne nécessitait que :

1. l'extension de la table des processus,

2. l'ajout d'un appel fork qui copiait le

processus en cours dans la partie de la mémoire dédiée à l'échange d'entrée/sortie déjà existantes, et faisait des ajustements dans la table des processus.

En fait, l'appel *fork* du PDP-7 nécessitait précisément 27 lignes de code assembleur. Bien sûr, d'autres changements étaient nécessaires dans le système d'exploitation et les programmes de l'utilisateur, et certains d'entre eux étaient plutôt intéressants et inattendus. Mais un mécanisme combiné de *fork/exec* aurait été considérablement plus compliqué, ne serait-ce parce que l'*exec* en tant que tel n'existait pas ; sa fonction était déjà réalisée par l'interpréteur, en utilisant des entrées/sorties explicites (*explicit I/O*). (Ritchie, 1980)

Les commandes *exit* et *wait*

Dans les systèmes Unix d'aujour d'hui, un processus qui a « fork()é » un nouveau processus peut faire un *wait()* pour attendre que le nouveau processus se termine. Ce dernier peut réaliser un *exit()* pour clore son exécution. Cette commande renvoie un statut « exit » pour signifier sa terminaison au processus original qui attend sur le *wait()*.

Comme pour *fork()* et *exec()*, les commandes *wait()* et *exit()* sont issues d'autres mécanismes. Mais dans ce cas, les mécanismes précédents n'étaient pas moins, mais plus sophistiqués.

Les primitives qui sont devenues *exit* et *wait* étaient considérablement plus générales que le mécanisme actuel. Un couple de primitives envoyait des messages d'un mot entre des processus identifiés :

Smes(pid, message)

(pid, message) = rmes ()

Le processus cible de *smes* n'avait pas besoin d'avoir une relation de filiation avec le receveur, même si le système ne donnait pas de mécanisme pour communiquer les identifiants des processus, à part le *fork()* qui renvoyait au père l'ID du fils et réciproquement. Les messages n'étaient pas mis en file d'attente ; le processus émetteur était retardé jusqu'à ce que le récepteur lise le message.

Le service de messages était utilisé ainsi : l'interpréteur *(shell)* parent, après avoir créé un processus pour exécuter une commande, envoyait un message au nouveau processus par *smes()* ; quand la commande se terminait (en supposant qu'elle ne tentait pas de lire un quelconque message) l'appel *smes()* bloqué de l'interpréteur renvoyait une erreur signalant que le processus cible n'existait pas. Ainsi le *smes()* de l'interpréteur devenait dans les faits un équivalent de la commande *wait()*.

Un protocole différent, qui exploitait la plupart des généralités offertes par les messages, était utilisé entre le programme d'initialisation et les interpréteurs pour chaque terminal. Le processus d'initialisation, dont il était convenu que l'ID était 1, créait un interpréteur pour chacun des terminaux, puis faisait un *rmes()* ; chaque interpréteur, après avoir lu la fin de son fichier d'entrée, utilisait la fonction *smes()* pour envoyer le traditionnel message « j'ai terminé » au processus d'initialisation, qui recréait un nouveau processus interpréteur pour ce terminal.

> Je ne me souviens pas d'autre utilisation des messages. Ceci explique pourquoi le service a été remplacé par les appels [*exit* et] *wait()* de l'actuel système Unix, qui est moins général, mais plus directement approprié au but recherché. Un bug évident dans le mécanisme explique aussi possiblement ce remplacement : si un processus de commande essayait d'utiliser des messages pour communiquer avec d'autres processus, cela pouvait perturber la synchronisation de l'interpréteur. L'interpréteur pouvait dépendre d'un message qui n'avait jamais été reçu ; si une commande exécutait *rmes(),* elle pouvait recevoir un faux message de l'interpréteur, et obliger l'interpréteur à lire une ligne d'entrée supplémentaire, comme si la commande était terminée. Si le besoin de messages généraux s'était manifesté, le bug aurait été réparé. (Ritchie, 1980)

Les outils de l'Unix PDP-7

Unix PDP-7 était non seulement un système d'exploitation polyvalent dont le noyau était compacté dans une mémoire de 4 000 mots, mais ses développeurs ont aussi créé un nombre important d'outils dont les descendants sont toujours utilisés aujourd'hui.

Dans son texte « Draft of the Unix Time-sharing System »[11] écrit en 1971 quand les deux systèmes Unix pour PDP-7 et PDP-11 existaient, Ritchie documente ces commandes et outils : *b*, *cat*, *chdir*, *chmod*, *cp*, *db*, *ed*, *ln*, *ls*, *mkdir*, *mv*, *nm*, *pr*, *rm*, *roff*, *sh*, *tm* et *un*. Seuls un petit nombre d'entre eux ne serait pas familier à l'utilisateur d'un Unix d'aujourd'hui :

- *b*, le compilateur B ;
- *db*, un débogueur pour des images mémoire (des images disque des processus qui ont planté) ;
- *roff*, un outil de traitement des documents, et précurseur de *nroff* ;
- *tm*, un outil pour imprimer des informations relatives au temps, en provenance du noyau (*kernel*), par exemple le temps dans le mode « utilisateur », le temps dans le mode noyau, le temps passé à traiter les interruptions ;
- *un*, qui liste les symboles non identifiés dans un exécutable.

Plusieurs outils d'Unix PDP-7 méritent d'être passés en revue de manière plus complète.

L'interpréteur (*shell*)

Unix PDP-7 a été le premier système d'exploitation à fournir un interpréteur de commande qui était modifiable et remplaçable par l'utilisateur. Des systèmes antérieurs comme CTSS et Multics avaient des interpréteurs en ligne de commande, mais ils étaient partie intégrante du système et ne pouvaient pas être changés ou modifiés.

Unix PDP-7 a été aussi le premier système à fournir à ses outils des abstrac-

11 Ritchie D.M., «Draft: The UNIX Time-sharing System», texte de 1971 publié sur le site The Unix Heritage Society [URL : http://www.tuhs.org/Archive/Distributions/Research/McIlroy_v0/UnixEditionZero-Threshold_OCR.pdf].

tions d'unité standard d'entrée (*standard input*) et d'unité standard de sortie (*standard output*). En substance, tout processus démarrait avec un fichier déjà ouvert pour lire des entrées, et un autre fichier ouvert pour écrire les sorties. Par défaut, ces fichiers étaient connectés au terminal de l'utilisateur.

L'interpréteur de l'Unix PDP-7 fournissait à l'utilisateur des mécanismes pour associer ces fichiers ouverts à des fichiers existants (ou nouveaux), et aussi aux fichiers spéciaux. Ritchie indique :

> La notation très pratique pour les redirections I/O, à l'aide des symboles « > » et « < », n'existait pas aux tout débuts du système Unix PDP-7, mais elle est apparue très tôt. Comme beaucoup d'autres choses dans Unix, cela a été inspiré d'une idée venant de Multics. Multics avait un mécanisme de redirection des I/O assez général (Project MAC, 1969) incarnant (représentant) des flux (*stream*) d'I/O identifiés qui peuvent être redirigés dynamiquement vers différents dispositifs, fichiers, et même à travers des modules spéciaux de traitement des flux. Dans la version de Multics qui nous était familière, il existait même une commande qui passait (orientait) la sortie suivante, normalement destinée au terminal, vers un fichier, et une autre commande pour rattacher la sortie au terminal. Si, sur Unix, on tape la commande :
>
> *ls > xx*
>
> pour avoir la liste des noms de fichiers dans [le fichier] xx, sur Multics la même opération était réalisée par :
>
> *iocall attach user output file xx list*
>
> *iocall attach user output syn user i/o*
>
> Même si cette séquence très maladroite était souvent utilisée à l'époque de Multics, et aurait été très simple à intégrer dans l'interpréteur de Multics, ça ne nous est pas venu à l'idée, ni à d'autres, à ce moment-là. Je suppose que l'ampleur du projet Multics en donne la raison : ceux qui ont implémenté le système I/O étaient aux Bell Labs à Murray Hill, alors que l'interpréteur a été réalisé au MIT. Nous n'envisagions pas de faire des changements dans l'interpréteur (c'était leur programme) ; réciproquement, les détenteurs de l'interpréteur n'avaient peut-être pas connaissance de l'utilité de *iocall,* sans parler de sa maladresse. (Le manuel Multics de 1969 (Project MAC, 1969) fait référence à *iocall* comme une commande *author-maintained,* c'est-à-dire non standard). Le système I/O d'Unix et son interpréteur étant sous contrôle exclusif de Thompson, cela a été l'affaire d'une heure ou deux pour l'implémenter, quand la bonne idée a fait surface. (Ritchie, 1980)

La notion de tube (*pipe*) est un autre concept important d'Unix qui n'arrivera pas dans le système avant la 3e édition de l'Unix PDP-11 en 1973.

Le traitement de texte avec *roff*

Mis à part la recherche sur les systèmes d'exploitation, le système Unix a été utilisé très tôt en tant que système de traitement de document. L'évolution de *roff*, l'outil de traitement de document,

est intéressante. Il a vu le jour sous le nom de *runoff*, l'outil de J. Saltzer, écrit en langage assembleur pour le système d'exploitation CTSS (Saltzer, 1965). Celui-ci a été réécrit par Doug McIlroy en BCPL pour le système d'exploitation Multics. Ensuite, il a été réécrit en assembleur PDP-7 pour donner *roff* (avec des fonctionnalités réduites) pour l'Unix PDP-7. Puis, pour justifier l'achat d'un mini-ordinateur PDP-11, les chercheurs sur Unix ont réécrit *roff* en langage assembleur du PDP-11 pour assurer les besoins en traitement de documents du département des brevets d'AT&T. Les utilisateurs courants de Linux considèrent peut-être que l'*open source* est un concept relativement nouveau, mais le partage et le développement de code source a été un mécanisme important depuis les débuts de l'informatique.

Le compilateur B

Comme nous l'avons vu plus haut, Thompson avait écrit un assembleur pour le mini-ordinateur PDP-7 pour faire du système Unix un système autonome. Le noyau et tous les outils originaux de l'Unix PDP-7 ont été écrits en assembleur PDP-7. Mais les développeurs Unix ont éprouvé le besoin d'utiliser des langages de haut niveau. Thompson indique que :

> Depuis le début [nous voulions écrire le système dans un langage de haut niveau]… C'était une influence de Multics. Vu la complexité qu'il y a à conserver l'existant, nous savions pertinemment qu'on ne pouvait pas conserver un élément [*a fortiori* en assembleur], l'écrire et le faire fonctionner, [alors] qu'il va forcément évoluer… PL/I [le langage de haut-niveau utilisé pour Multics] était de trop haut niveau pour nous, ainsi que la version plus simple de PL/I dans Multics (un truc appelé EPL). Après le montage [d'Unix PDP-7], ou en même temps que sa sortie, BCPL était en train d'émerger et c'était un ticket gagnant pour nous deux [*i.e.* Thompson et Ritchie]. Nous avons été tous les deux captivés par le langage et nous avons beaucoup travaillé avec[12].

BCPL était un langage pour systèmes « orientés-mot » développé par Martin Richards à l'Université de Cambridge (Richards, 1969), qui était conçu pour être assez portable sur d'autres machines et systèmes. Comme BCPL était cependant encore un trop « gros » langage pour le PDP-7, Thompson a repris les structures essentielles de BCPL et a écrit un interpréteur d'un langage intermédiaire implémentant ces structures pour le PDP-7. Il a ensuite écrit un compilateur à destination de ce langage intermédiaire, et a créé le langage B.

> C'était le même langage que BCPL, mais il avait l'air complètement différent. En termes de syntaxe c'était une réécriture, mais la sémantique était exactement la même que BCPL. Et en fait, la syntaxe était telle que, si vous

12 Entretien de M. Mahoney avec Ken Thompson, 1989 [URL : https://www.princeton.edu/~hos/mike/transcripts/thompson.htm].

> ne regardiez pas de trop près, vous auriez dit que c'était du C. Parce que c'était du C, mais sans les types[13].

Très peu de choses de cette phase d'Unix ont survécu, seulement quelques programmes PDP-7 simples, écrits en B. Le langage B, et son compilateur, allaient évoluer en de nombreuses étapes vers le langage C (Ritchie, 1993). Ritchie et Thompson réécriront le noyau d'Unix en langage C de haut niveau en 1974 (Ritchie & Thompson, 1974), atteignant leur objectif d'écrire le système en langage de haut niveau.

Restaurer le système Unix PDP-7

Pendant de nombreuses années, le système Unix PDP-7 est resté essentiellement une créature mythologique. Même si son existence était documentée[14] (Salus, 1994), aucun code source n'avait subsisté à part pour la commande *dsw*, posté par Dennis Ritchie sur le groupe de nouvelles net.unix-wizards de Usenet en 1984.

En tant que fondateur de l'Unix Heritage Society[15], j'ai passé beaucoup de temps à récupérer des vieux systèmes Unix et les remettre en fonctionnement. Les restaurations de la première version de l'Unix PDP-11 et du premier compilateur C (Toomey, 2009) ont été des succès notables. Mais le système Unix PDP-7 restait insaisissable.

En 2016, un membre de longue date de l'Unix Heritage Society a révélé qu'il était en possession d'un tirage papier du code source d'Unix PDP-7, qu'il avait copié dans les années 1980 lorsqu'il travaillait aux Bell Labs d'AT&T. Nous l'avons encouragé à le scanner et à le partager avec la Society.

Le code source de n'importe quel système informatique est inutile en soi, à moins qu'il n'y ait un environnement pour le convertir en exécutables pour la machine et un environnement qui puisse faire tourner ces exécutables. En 2016, le PDP-7 était un souvenir lointain et, même si une telle machine était disponible, il n'y avait pas d'assembleur pour convertir le code source en format exécutable.

Heureusement, Bob Supnik et une cohorte d'excellents développeurs avait conçu SimH (Supnik & Walden, 2015), un simulateur de nombreux ordinateurs anciens incluant le PDP-7 et ses périphériques. Lorsque le code source de l'Unix PDP-7 a été rendu disponible, une équipe de trois personnes (Phil Budne, moi-même et Robert Swierczek) a commencé à convertir le code source en un système fonctionnel.

La première tâche consistait à écrire un nouvel assembleur PDP-7, ce qui a été

---

13 *ibid.*

14 « Draft : The UNIX Time-Sharing System », D.M. Ritchie, 1971. *Cf.* note 11.

15 Site web de l'association [URL : http://www.tuhs.org].

initié par moi-même, et complété ensuite par Phil Budne. Nous avions le code source pour l'assembleur Unix PDP-7, mais nous étions dans une situation de type « l'œuf et la poule » : la source de l'assembleur ne pouvait s'assembler lui-même.

Avec notre assembleur, nous avons pu assembler le code source en code machine pour le PDP-7 mais nous ne pouvions pas encore exécuter ces instructions. Nous ne pouvions pas utiliser SimH, parce qu'il simulait un système entier ; pour exécuter un code Unix, nous avions besoin d'un système complet : machine, noyau, un système de fichier opérationnel et des outils. Un bug dans une seule de ces parties empêcherait le système de fonctionner correctement.

À la place, nous avons choisi d'implémenter un simulateur du mode *User* : un mode qui exécute la plupart des instructions machine du PDP-7, et convertit les appels système de l'Unix PDP-7 en appels système à l'OS hôte sous-jacent. Wine (Amstadt & Johnson, 1994) est un exemple de simulateur de mode *User* similaire. Utilisant ce simulateur, nous pouvions tester notre assembleur et le code source des outils originaux de l'Unix PDP-7 sans se soucier du noyau Unix ou du système de fichiers.

Avec un assembleur et des outils en lesquels nous pouvions avoir confiance, nous nous sommes intéressés au noyau de l'Unix PDP-7 et à la construction d'un système de fichiers. Phil Budne s'est chargé de faire fonctionner le noyau, et j'ai écrit un outil pour construire un système de fichiers approprié. Les deux missions étaient ardues. Les développeurs d'Unix avaient laissé très peu de commentaires dans leur code assembleur, et les instructions PDP-7 étaient étrangères à tous les membres de l'équipe. Nous avons passé beaucoup de temps à déduire l'objectif de sections de code assembleur, et à ajouter des commentaires et annotations à celui-ci. En ce qui me concerne, le manque total de documentation sur le système de fichiers en a compliqué le développement. En utilisant les informations que nous avions à portée de main (le code source du noyau, la documentation du système Unix PDP-11 et la possibilité de faire exécuter pas à pas les instructions sur SimH), j'ai pu construire au final un système de fichiers Unix PDP-7 opérationnel.

Phil Budne a eu des difficultés analogues avec le noyau de l'Unix PDP-7. Il n'y avait pas de documentation sur l'écran Graphics-2, qui avait été construit au sein d'AT&T[16]. Non seulement Phil devait déduire ses modes opératoires, mais il devait aussi ajouter du code à SimH pour simuler le terminal.

Ken Thompson avait pris un mois en 1969 pour écrire le noyau, l'assembleur, l'interpréteur et le débugger. Sur ces quatre outils essentiels, le code source de l'interpréteur avait disparu. Utilisant

16 *Cf.* la page dédiée sur le site des Bell Labs [URL : http://cm.bell-labs.co/who/dmr/spacetravel.html].

les informations fournies par Ritchie dans son article « The Evolution of the Unix Time-Sharing System » (Ritchie & Thompson, 1974) et les bribes du code source PDP-7 existant, Phil Budne a reconstruit héroïquement un interpréteur fonctionnel pour le système Unix PDP-7.

Lorsque toutes ces tâches réalisées ont été mis bout à bout, l'équipe a pu annoncer que le système temps-partagé Unix PDP-7 avait été complètement remis en fonctionnement ; le code source original et les outils utilisés pour la reconstruction sont disponibles en téléchargement sur Github[17]. Cela inclut plusieurs outils pour lesquels le code source avait disparu : *cp*, *ln*, *ls* et *mv*.

En arrière-plan, Robert Swierczek avait travaillé à la reconstruction du compilateur B. Son code source avait également été perdu, mais l'interpréteur pour le langage intermédiaire avait survécu. Robert a utilisé le code source du plus ancien compilateur C, retiré le code concernant les types, et réorienté pour produire le code de l'interpréteur. Chef-d'œuvre d'amorçage (*bootstrapping*) inversé, Robert a écrit le compilateur B pour qu'il soit valable pour du code B ou C, et pour qu'il puisse se recompiler lui-même. Pour construire le compilateur, il est d'abord compilé en utilisant un compilateur C moderne. Ce compilateur qui n'est pas encore un compilateur PDP-7, est ensuite utilisé pour compiler le code source du compilateur B (une nouvelle fois), générant la version pour le langage intermédiaire qui peut être exécutée en utilisant l'interpréteur B de l'Unix PDP-7.

Conclusion

Le développement d'Unix a été en quelque sorte une réaction à l'encontre de la mentalité « *bigger is better* » de Multics. Thompson avait l'intuition que « *c'était une bonne idée de sortir de Multics. C'était trop gros, trop cher, trop sophistiqué. Il était clair que c'était un exercice de construction d'usine à gaz.* »[18]

Mais Unix n'a pas été un simple rejet du concept de Multics : en effet, beaucoup d'idées de Multics ont été simplifiées et ajoutées à Unix. Ritchie souligne :

> Nous étions un peu oppressés par la mentalité « grand système ». Ken voulait faire quelque chose de simple. Unix n'est pas qu'une réaction contre Multics, c'était une combinaison de ces choses. Multics n'était plus là pour nous, mais nous appréciions le sentiment d'informatique interactive qu'il procurait. Ken avait des idées à creuser sur comment faire un système. Le matériel disponible ainsi que nos penchants

17 *Cf.* la page dédiée sur la plateforme de partage de code Github [URL : https://github.com/DoctorWkt/pdp7-unix].

18 Entretien de M. Mahoney avec Ken Thompson, 1989 [URL : https://www.princeton.edu/~hos/mike/transcripts/thompson.htm].

> nous ont incité à construire des petites choses élégantes plutôt que des choses grandioses.

Dans le temps, le système d'exploitation Unix allait épouser une philosophie de conception qui a été résumée par Doug Mcllroy :

> C'est la philosophie Unix : écrire des programmes pour faire une chose et la faire bien. Écrire des programmes pour travailler ensemble, collaborer. Écrire des programmes pour manipuler des flux de texte, parce que c'est une interface universelle (Raymond, 2003).

Avec la version PDP-7 d'Unix, Thompson, Ritchie et d'autres expérimentaient encore les concepts et structures qui allaient se développer en fin de compte et se cristalliser dans cette philosophie. Ils étaient aux prises avec la structure du système de fichiers, les mécanismes fondamentaux des processus Unix, les caractéristiques de l'interpréteur et la possibilité d'abstraire les opérations sur les périphériques en des opérations sur des fichiers génériques. Les deux derniers éléments qui allaient donner cette philosophie de « boîte à outils » à Unix, les tubes et le noyau portable, allaient arriver dans les quatre années suivantes.

En utilisant une plateforme matérielle dépassée, le PDP-7, et avec seulement 8 192 mots de mémoire, Thompson et Ritchie ont pu construire un système d'exploitation multitâches, multi-utilisateurs avec un système de fichiers multi-répertoires. L'Unix PDP-7 n'a pas seulement distillé et simplifié beaucoup de concepts d'autres systèmes (Multics, Project Genie), il a aussi introduit des innovations comme les liens physiques, les périphériques représentés comme des fichiers spéciaux et les redirections généralisées d'entrée/sortie. Beaucoup des caractéristiques, outils et appels système introduit dans l'Unix PDP-7 ont cours aujourd'hui encore dans les descendants comme BSD[19], et dans des réécritures très propres comme Linux. L'héritage de ce tout petit système d'exploitation vit encore, cinquante ans après sa création.

19 Berkeley Software Distribution – une famille de systèmes d'exploitation dérivés d'Unix, produits originellement à l'Université de Berkeley.

Bibliographie

Amstadt B. & Johnson M.K. (1994). « Wine ». *Linux Journal,* Issue 4, p. 3.

Brooks F.P. (1975). *The mythical man-month: essays on software engineering.* Reading, Mass. : Addison-Wesley.

Corbato F.J., Saltzer, J.H. & Clingen, C.T. (1972). « Multics: the first seven years ». *American Federation of Information Processing Societies: AFIPS Conference Proceedings,* 1972 Spring, Joint Computer Conference, Atlantic City, NJ, USA, May 16-18, pp. 571-583.

Nyman L. & Laakso M. (2016). « Notes on the history of fork and join ». *IEEE Annals of the History of Computing* 38(3), 84-87.

Project MAC (1969). *The Multiplexed Information and Computing Service: Programmers' Manual.* Cambridge MA. : Mass. Inst. of Technology.

Raymond E.S. (2003). *The art of Unix programming.* Reading : Addison-Wesley Professional.

Richards M. (1969). « BCPL: A Tool for Compiler Writing and System Programming ». *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference, AFIPS '69*, New York : ACM, pp. 557-566

Ritchie D.M. (1980). « The Evolution of the Unix Time-Sharing System ». *Proceedings of a Symposium on Language Design and Programming Methodology,* London, Springer-Verlag, pp. 25-36.

Ritchie D.M. (1993). « The Development of the C Language » in The Second ACMSIGPLAN Conference on History of Programming Languages, HOPL-II, New York, ACM, pp. 201-208.

Ritchie D.M. & Thompson K. (1974). « The UNIX Time-sharing System ». *Communications of the ACM* 17(7), pp. 365-375.

Saltzer J.H. (1965). *Manuscript Typing and Editing,* 2nd edition, Cambridge, Mass. : MIT Press, p. AH.9.01.

Salus P.H. (1994). *A Quarter Century of UNIX.* New York : ACM Press/Addison-Wesley Publishing Co.

Supnik B. & Walden D. (2015). « The Story of SimH ». *IEEE Annals of the History of Computing* 37(3), pp. 78-80.

Toomey W. (2009). « The Restoration of Early Unix Artifacts » in 2009 USENIX Annual Technical Conference (USENIX ATC 09), San Diego, USENIX Association.

Témoignage / *Testimony*

Unix: A View from the Field as We Played the Game

Clement T. Cole
Intel Corporation.

Abstract

UNIX is a classic example of a "Christensen Disruptive Technology." It was a cost-effective solution, produced at the right time, built by researchers at AT&T for themselves, and was not originally considered seriously by its competition. The UNIX Operating System had simple goals. It ran on modest hardware, and was freely shared as a result of AT&T legal requirements. As a result, a new computing customer developed, a different one than was being targeted by the large firms of the day. UNIX was targeted at the academically-inclined; it was economically accessible, and since its Intellectual Property (IP) was published in the open literature and implementation was available to the academic community fundamentally without restriction, the IP was thus "free" and able to be examined/discussed/manipulated/abused by the target users. While its creators wrote UNIX for themselves, because they freely shared it with the wider community, that sharing fed on the economics in a virtuous circle as this community developed into a truly global one. I will trace a little of the history of a small newsletter to today's USENIX Association and some of its wider social impact.

Keywords: UNIX, history of UNIX, operating systems, open system, disruptive innovation.

A Brief Personal History

In the mid-1970s I was a student at Carnegie Mellon University (CMU), studying Electrical Engineering and Mathematics. I worked for the University as a programmer. Since that time, I have secured a number of degrees there and at other institutions and have since spent the next 40 years developing computing systems. I have been lucky enough to be part of some exciting projects: from core UNIX development, one of the first IP/TCP implementations, to many other technologies. I have published papers and secured patents in computing since that time and currently lead a team of engineers building supercomputers.

A Christensen Disruptive Technology

I contend that UNIX is a classic example of a "*Christensen Disruptive Technology*" (Christensen, 1997). It was a cost-effective solution, produced at the right time, built by research computer scientists at AT&T for an unserved or under-served market (themselves – who were computer programmers), which at its birth was not seriously considered by its competition. In those days, computers were designed to execute either high end scientific applications typically at the large national/government research laboratories or to perform so called "back-office" work traditionally supporting finance and accounting at large commercial business firms.

The point is that when UNIX was originally written, the purchasers of computing equipment were not primarily made up of the actual programmers of the computing system. Thus, the major computer firms tended to ignore the needs or desires of those the programmers, as they were not seen by the larger firms as the "true" customer since they did not directly pay the bills. However, the target consumers of the UNIX system were also programmers, and of course they did see the value (UNIX was "good enough" for themselves), and a community grew up around it because UNIX was a cost-effective solution for them to do their own work.

UNIX had simple goals, ran on "*modest hardware of the day*" (Ritchie, 1974) and was freely shared as a result of AT&T legal requirements (Pinheiro, 1987). The fact is UNIX might not have been successful if it had required much larger equipment to execute. A smallish DEC PDP-11/40 class system that a traditional UNIX system ran such as an PDP 11/34 with max memory (256K bytes) would just barely suffice, but that cost on the order of $50K-$150K after disk, tapes, etc. (1977 dollars). If you wanted a PDP 11/70 class system that could address as much as 4M bytes and offer many more services, it was closer to $250K[1] (1977 dollars). To help the reader scale to modern times, the cost of a graduate researcher might have been about $5K-$10K.

1 For comparison $50K/$150K/$250K 1977 dollars is $208K/$622K/$1M 2017 dollars.

Similarly, while these prices may seem large to today's developer using a "personal computer" that costs $500-$1000[2] (in 2017 dollars), another key point is that in those days you did not own the hardware yourself, a user (programmer) like me, used a system owned/operated by someone else; typically owned by your employer or the university you attended. In comparison, the large systems of the mid-1970s that were installed to support scientific or back-office style work usually began at $500K and often reached multiple millions of dollars – so the UNIX systems at $50K-$100K really were modest (much less counting for inflation of the dollar or euro to today's prices).

The 1956 AT&T consent decree

The question is how did this new community occur in the first place? I previously mentioned that UNIX was "*freely shared as a result of AT&T legal requirements*." I am referring to the 1956 AT&T consent decree. This order had extremely important side effects for those of us in the computer business. It is well explained in detailed legal wording by John Pinherio in a paper published in the *Berkeley Technical Law Journal* (Pinherio, 1987), although it is a bit difficult to follow. Instead, consider a quick quote from Wikipedia on the history of AT&T here as this prose is directly to my point and easier to understand:

> In 1949, the Justice Department filed an antitrust suit aimed at forcing the divestiture of Western Electric, which was settled seven years later by AT&T's agreement to confine its products and services to common carrier telecommunications and license its patents to "all interested parties." A key effect of this was to ban AT&T from selling computers despite its key role in electronics research and development. Nonetheless, technological innovation continued.

My non-legal description of the decree is that in return for granting AT&T a legal monopoly for the phone business in the USA, AT&T had to agree to a number of behaviors. One of them was that they were not allowed to be in the computer business[3], but the other was that all AT&T had to agree to continue to work with the academic research community and industry at large as it had done in the past, and must make all of its inventions available to the academic community at no charge or by licensing them for "*fair and reasonable terms*" to their industrial partners – all of those licenses were monitored by the US government.

From a historical standpoint, and as a result of the decree, the electronics industry got a major boost with another invention from AT&T: the transistor.

2 Similarly, $500/$1000 in 2017 dollars becomes $120/$241 in 1977 dollars (see [URL: http://www.dollartimes.com] for reference).

3 They were also protected, as other firms such as IBM was not allowed to compete with AT&T in the phone business either.

While it was invented in 1947 in Murray Hill at Bell Labs, clearly it was firms such as Fairchild Semiconductor, Texas Instruments (TI), Intel, etc. that would make the money on its invention. We as consumers and as a society clearly have benefited greatly. AT&T was required to license the device (the transistor) to anyone, and they did. In fact, because of the decree, AT&T had created an office in Murray Hill (New Jersey) called the "Patent and Licensing Group" whose sole job was to write those licenses for firms that inquired and asked for them. Ironically this is how UNIX got its start, as the word processing system for those same people. Indeed, a computerized help made a great sense for them due to the need to rewrite and reformat those sometimes complicated licenses.[4] (Ritchie, 1984).

The key point here is that by the late 1960s, early 1970s when the computer science community was made aware of the UNIX technology, AT&T was required by law to license its technologies to everyone that asks for them and actually had created and instituted the processes and procedures to do just that. By the original legal definition, the 1956 consent decree had made UNIX "open", but licensed. This would really be indeed "Open Source" as we think about it today. This is an important point I will come back to later in the paper.

4 See also: "Dennis Ritchie obituary", Martin Campbell-Kelly, The Guardian website, 13th October 2011 [URL: https://www.theguardian.com/technology/2011/oct/13/dennis-ritchie] (accessed 201709041527EDT).

At the time, doing computing research made perfect sense for AT&T, given their core business (telephony), since the "heart" of a telephone system was in fact a stored program, digital computer. In fact, we can look at the Bell System, the phone switching network, as the world's largest and most complex computer system. But by statute, AT&T is not allowed to be in the (formal) computer business. Also, as a side product of building the phone network, just like the transistor which was another core technology created by the researchers at AT&T, software and algorithms were being developed. Of course, the research in software and algorithms would lead to UNIX.

The Murray Hill team had then (and still has) many researchers with degrees in core science and technology such as mathematics, physics, and other academic fields who continue to publish papers about their ideas in the open literature. Those ideas were quite different from other computer systems being discussed at the time in the same places and journals. The same researchers developed the code and ran it internally for their own use; just like they built transistors and used them, their research was also "applied" or in patent terminology: "reduced to practice" from theory.

One of the groups in Murray Hill was a computer science research group, with a number of its members working on topics such as operating systems, compilers and languages, and similar techno-

logies. In the mid-1960s, a number of members of this team had been working jointly with MIT and General Electric on a large-scale computing utility system, the Multics system which was part of project MAC[5]. Two of the members from AT&T were Ken Thompson and Dennis Ritchie. AT&T abruptly stopped working on the project. Ken and Dennis were operating system researchers so it is no surprise when AT&T left project MAC, they wanted to continue working on OS topics at Murray Hill.

Another research group in Murray Hill had been doing speech work using a PDP-7. While Ken was not in that specific group, he managed to borrow their PDP-7 when they were finished with their original experiment since Ken's team's proposal to purchase their own computer for research had been denied. Later, when the first versions of UNIX started to show promise when running on the PDP-7, Ken's group had to scrounge the $60,000 needed to purchase their first PDP-11 that they would use for the next phase of their work (Ritchie,1984). This work was after all research, prototyping, and exploration of ideas.

By 1974, when Dennis and Ken published the original UNIX paper in CACM (Ritchie, 1974), AT&T researchers had developed a technology their employeer was not allowed to directly sell, and in fact were required to make available to "*all interested parties*". But because they had published about their ideas (and thus the technology itself), the technology and the ideas behind it, drew interest outside of AT&T. Quickly the academic community started to ask about it. By the rules of the 1956 consent decree, AT&T was required by law to make UNIX available to people asking about it. The Murray Hill Technology license office did the same with the fee being a $150 tape copying charge. The fee covered what it cost AT&T to purchase, write and mail the tape back to you. The key point is that if you worked at an academic institution, it was extremely easy to get a license for UNIX for your institution and copies of the UNIX implementation with full sources from Ken and Dennis and many, eventually most, did.

As a result, a new and unexpected computing customer started to develop, which was different from what was being targeted by the large commercial computer firms and valuing different features from these traditional computing systems. This new type of computer customer, of which I was one, did not care as much about what those large systems could do, as their internal core IP of the big systems were not "freely" accessible to us and it did not actually serve us as well UNIX did in many cases. In addition, with UNIX we could do a lot with what we had and the core IP was freely available to us (open), so we could (and would) enhance it as we desired.

5 "CSAIL Mission and History", CSAIL [URL: http://www.csail.mit.edu/about/mission-history] (url accessed 20171153014O2EDT).

The UNIX Community Emerges to Share

As I mentioned, anyone who asked could license and obtain code from AT&T and use it. In fact, the term we used was that AT&T actually "abandoned" the sources at your front door. "*There was no warranty of any kind. You asked for it, there it was. You figure it out.*" The first to ask was Prof. Lou Katz of Columbia University. Ken made a copy of the contents of what we would later call the Fourth Edition available to him[6]. This exercise would be repeated many times over the next few years.

I cannot express the importance of the abandonment comment enough. AT&T senior management really did not have a desire that the actions of the company be seen by the US courts as being in the computer business so they did not want to have anything to do with you as a customer after they delivered the technology. As UNIX users, we were all fellow travelers; researchers doing what we wanted with this technology. Thus, AT&T actions encouraged us (as customers and licensees) to work with each other. Furthermore, the core AT&T research published about UNIX, using the sources and the intellectual property that their employees had developed continued to encourage us to do the same: research and publish about the same technology. This is exactly the same behavior we would later put a name to – we call it "open source."

Similarly, at the time of UNIX started to be available outside of AT&T, Digital Equipment (DEC) was releasing its own operating systems for the PDP-11: originally, DOS-11 and RT-11; which were followed by RSX-11 and RSTS[7]. As the manufacturer of the PDP-11 hardware, DEC was hardly encouraging its user base to use UNIX. They wanted their customers to use their technology and purchase licenses for their software products. They charged monetary fees for the use of the software such as the compilers and other layered products, which was a manner in which these firms made money. By our use of UNIX, we were customers of their hardware, but not of their software products. UNIX was free of real direct monetary cost to us as academics; hence UNIX became competition to DEC software products even if it was not a formal product[8].

6 Private Communication, although this has been discussed and can be been verified elsewhere with a modern search engine.

7 "Index of /pdf/dec/pdp11" [URL: http://bitsavers.trailing-edge.com/pdf/dec/pdp11/] (access 201709041612EDT).

8 I should point out there were eventually some fees for the commercial use of UNIX. In comparison to the cost of the hardware at the time, those fees were originally a few thousand dollars, and actually not much different than the fees DEC charged for its OS. In the case of UNIX a licensee also got the sources from AT&T. I'm going to ignore this part of the case for this argument because I personally never found those fees to actually be more than a small hurdle in practice. I was working in the industry from the late 1970s onward, and had installed a number of commercial UNIX sites, and had obtained the first commercial license for a University at CMU in 1978. The fact is that the cost of a PDP-11 or VAX hardware dominated the cost of a UNIX installation in the 1970s and 1980s.

Another force was clearly at work. AT&T was legally prohibited from supporting us as customers, and DEC did not want to provide support for the UNIX technology either because the software we had chosen to use (UNIX) was competing with its own; thus, we as users, quickly wanted and needed to talk to each other to share solutions to common problems. I observed that communal activity was forced upon us because it would not have been possible to exist economically otherwise. One thing that was becoming increasingly obvious by this time was that the UNIX OS used the hardware differently (better) than the DEC software did, so it would often reveal hardware errors that the DEC standard software would not. A shared approach when working with the hardware vendor was important. If DEC field service was getting the same complaints from the Massachusetts Institute of Technology (MIT), Carnegie Mellon University (CMU), Harvard University, much less AT&T, there was a better chance we were listened to as a joint voice. That is to say, by acting as a community with common goals, not as individual users, we had more power over our key vendors, of which DEC was the most important.

Within the USA when UNIX was starting to spread to many sites, some tier-1 research and academics institutions had access to ARPANET[9] as part of their work with the US Government. Some of the primary Computer Science departments at the major research institutions that had received a copy of UNIX from AT&T were, of course, connected via email on it, but not all. The University of California, Berkeley (UCB) who would become so important in the UNIX story later, was not yet part of the ARPA community. But we all did have physical mail, we all read the same journals and published in the same conferences.

By the time the Sixth Edition was released, Ken added the following note to the UNIX installation (start up) document:

> – send your name(s) and address to:
> Prof. Melvin Ferentz
> Physics Dept.
> Brooklyn College of CUNY
> Brooklyn, N.Y. 11210

This mailing list would be called the *UNIX News*. It was published when Mel had enough information. In fact, some copies of some of these newsletters can be found at the USENIX Association web site today[10]. In time, the members of the mailing list started to meet regularly, originally in New York City. People came together and someone might talk about a new trick, or optimization. Sometimes Ken or Dennis even visited and brought a patch or two and they all often brought

9 Ancestor to the Internet.

10 *USENIX Notes* April 2010 [URL: https://www.usenix.org/legacy/publications/login/2010-04/openpdfs/usenixnotes1004.pdf] (accessed 201709041929 EDT).

their own tweaks and additions. It was all very informal and collegial.

By the time of the Sixth Edition was released by AT&T, it was clear that UNIX was a hit in the academic community, although the commercial computing business hardly had noticed it. The number of licenses that were assigned at this time was probably around 50-100, but that number would grow exponentially shortly. It had become the largest licensed item that the Patent and License Group had ever seen. The rules that the license declared had started to become more straightforward and a bit more formal: this was after all, AT&T's Intellectual Property (IP) and needed to be treated as such, by its licensees. Similarly, the groups of like-minded users started to come together all over the world.

Licensing and Code Access

Almost all software has some sort of license for its use and care associated with it. The UNIX license said the IP was owned by AT&T and we could use it and share it with other licensees – which meant that information could and would be exchanged freely just as it is today in the so called "open source" movement.

The standard practice of the time was that someone at each site would send a photocopy of the "signature page" of their AT&T license to some other site to demonstrate that your site had indeed signed the AT&T license agreement. At Carnegie Mellon University (CMU), we kept a file of the collected signature pages that we received from places like MIT, UCB, Harvard, Purdue University. UCB would have done the same, as would have MIT. The key point being that with such a piece of paper in place, the actual code then flowed between the sites quite easily and without restrictions.

One thing that was interesting and was different at each institution was the handling of source protections (i.e. which human users had access to the sources) — the core UNIX IP itself. At CMU, those of us with need for access to the UNIX sources (i.e. when we took the undergraduate OS course) had been required to sign a one-page, sub-license with the University stating we understood it was AT&T IP's and we would guard it appropriately. I have not heard of any other school doing something similar. On the other hand, once you had signed that document, you were added to a group that had access to the code and in an extremely free and unregulated manner. The impressions I have had at most other institutions such as MIT or Purdue was that if we had a reason to need access, it was granted in some manner.

When I have discussed this idea with some of my peers that came in the UNIX community, by the mid-1980s, early 1990s when UNIX moved from the DEC based PDP-11 and VAX systems to ones made by firms such as Masscomp, Sun and the like, it seems that for many

students it was somewhat hard to obtain direct access to the UNIX sources: you had to be part of the closed "UNIX club" to obtain access to the AT&T IP (i.e. AT&T provided source code itself). Certainly, from a source code standpoint, this seems to have been particularly true at the largest educational institutions (at least of ones that I am aware in the USA, or so it has been expressed to me by people who I trust that lived in those times).

However, I counter the "closed club" argument with two points. First, the IP (the ideas) had been completely published by this time, although the source code had not been. The AT&T owned source was never the definition of UNIX; it has always been the ideas of how to build a system were and are the definition of UNIX, which I will discuss more in a minute. Second, for me, having grown up studying UNIX 15-20 years earlier as part of my formal education, using it for work, etc., I never found UNIX to be anything but freely available. However, I can see that if I had been at an institution that had been running UNIX (particularly a binary only implementation) and I had not been given access to the sources (as had been my experience working in the industry), it could have been considered "forbidden fruits" for those people even though the truth was the core UNIX IP (the ideas) really was not.

Teaching UNIX and the Lions' Text

The important thing that had happened by the mid-1970s that is extremely interesting was that the academic community began to study and teach lessons from UNIX. Previously, many schools had developed courses that used a "toy operating system" to teach. With the arrival of UNIX, a real OS could be examined, discussed and understood. In fact, one professor, John Lions of the University of New South Wales, wrote a booklet called The *Lions Commentary on the UNIX 6th Edition* for his students taking courses 6.602B and 6.657G[11]. He included the sources to the UNIX kernel in his booklet (Lions, 1977). This document itself was widely circulated as a bootleg photocopy and would become cherished by the UNIX community. The author still has his own nth generation xerographic copy from that time.

Clearly, the *Lions*' students, much like me, were immersed in the code and had access to the AT&T IP. The fruits were hardly forbidden. But his book does highlight a new issue that would become troubling for the UNIX community years later – the idea of the AT&T IP being a trade

11 The author still considers it the best treatise on how a modern OS works – although the MIT folks have updated it to use the Intel x86 architecture and a modern C compiler since sadly many current programmers no longer find the PDP-11 understandable. See "Xv6, a simple Unix-like teaching operating system", by Russ Cox, Frans Kaashoek and Robert Morris [URL: https://pdos.csail.mit.edu/6.828/2014/xv6.html] (accessed 201709031127EDT).

secret. AT&T controlled the publishing of the Lions' text because the book had the sources to UNIX printed in it. However, later books like *Design of the UNIX Operating System* (Bach, 1986), *Design of and Implementation of the 4.3BSD UNIX Operating System* (Leffler, 1989), and later *The Magic Garden Explained : The Internals of UNIX System V Release 4 An Open Systems Design* (Goodheart, 1994) which described UNIX as well, were not controlled. To get a copy of the Lions text a licensee had to obtain it from AT&T's patent license office and only licensees could do so – hence the tendency to photocopy them.

A key point is that by the time many of us left university in the late 1970s, we had been "mentally contaminated" by the AT&T IP – the core UNIX ideas from Ken, Dennis and the team in Murray Hill. Their ideas were now how we thought; they were the foundation of how we built systems, and led to how we would teach future programmers.

Absence of support and the USENIX Association

As I have said, when the UNIX community began to take shape, because of the consent decree and the fear of another anti-trust case against them, the AT&T management had officially and specifically abandoned us as UNIX licensees from a legal standpoint. The truth is that Ken and Dennis and the rest of the team in Murray Hill, NJ were wonderful people. They did want to help us as users of the technology and fellow members of the greater UNIX community, so when they could, they did. They were our friends and colleagues and we helped them when we could, too. But they were limited in what that could do. As it happened, Ken had a set of patches to the Sixth Edition he wanted to get out to the community to fix a number of issues that had arisen since the original distribution. There was no formal way to do it. Officially, such an update distribution was not allowed ("*no warranty implied*" said the license we had signed) and there really was not yet a formal mechanism even it had been allowed by the AT&T license.

At the time, Ken was driving his family to California for a sabbatical at UC Berkeley. He made a stop at the University of Illinois to see a soon-to graduate student, the late Greg Chesson who was about to start to work full time at Bell Labs. Somehow Ken's patches managed to get copied to a number of sites on the ARPANET. (I never knew how and Greg is not here to tell us). I know we got them eventually at CMU, and I believe MIT got them around that same time. There was a comment about the patches existing in the *UNIX News* and by then they were being passed by tape when different users got together. Simply put, the UNIX community was sharing what it had with each other. We were all in it together.

The meetings, newsletters, sharing of tapes were really the beginning of

the USENIX Association[12] (USENIX, 2017). UNIX News started what became a series of conferences, in those days twice a year, winter and summer, when the UNIX users could come together[13]. The benefits of the conversations beside basic fellowship of course was we could share knowledge in person and specifically share implementations, additions and corrections. We would organize talks on different topics and to many of us the highlight of the meetings was the creation and redistribution of a 9-track magnetic tape which held the collected offering from different sites. Archives of these distributions still can be found today in places such as the UNIX Source archives maintained by the UNIX Heritage Society[14].

Soon the conferences, called the semi-annual USENIX conferences, were not only in NYC. They were also in Cambridge (Massachusetts, USA) at Harvard University, then in California, Toronto and Boulder. But the idea was not just US centric, the conferences started to spread to Europe, too. The UK was first, with Cambridge (UK), but quite quickly, the continent got busy. A European UNIX Group formed. The USENIX Association started to publish proceedings of its conferences, and instead of having just two conferences a year, they started to sponsor special conferences on topics within the community. Given the academic seeds, the USENIX Association itself became a well-respected publishing arm and its conferences became preeminent, i.e. publishing papers at USENIX conferences was important for getting tenure if you were a Computer Science person at one on the top Universities. To this day, USENIX sponsored conferences, such as the FAST and the Security conferences, are the top conferences in those specific areas to publish for storage or security topics.

In fact, one of the most important gifts to the computer community that I can think of came from the USENIX Association, what we call the "open access" movement. Starting in 2008, when I was on the board and serving as the president of the USENIX Association, the open access movement was born to make sure sharing of information was easy[15]. The concept of open access simply put is that ideas need to be open to fuel creativity and experimentation and, thus, more and better ideas. As a result of this change, all of USENIX's publications have been available to anyone to read without any fees, which stems directly from the original UNIX philosophy, ideas and actions.

12 "About Usenix", USENIX Association [URL: https://www.usenix.org/about] (accessed 201709041858 EDT).

13 The newsletter itself was renamed ;*Login* which was a credit to the original UNIX herald and continues to be published under than name.

14 "The Unix Heritage Society" [URL: https://www.tuhs.org] (accessed 201709041904 EDT).

15 USENIX Notes April 2010 [URL: https://www.usenix.org/legacy/publications/login/2010-04/openpdfs/usenixnotes1004.pdf] (accessed 201709041929 EDT). USENIX Update April 2012 [URL: https://www.usenix.org/blog/usenix-supports-open-access] (accessed 201709041930 EDT).

Open, Free, Libre and Gratis

Parts of the original UNIX source code and its descriptions of how it worked were published in journals, papers and books. At conferences, such as USENIX's, the sources themselves used to build the entire system were being "freely shared." Note that the description of the code was open. While you never did need to have an AT&T license to come to a USENIX conference, those that came, were covered by their employer or academic institution, at least initially.

Thus, we had USENIX conferences or equivalents in Europe and we used them to trade code back and forth. The manner in which we traded code was often physically carrying or mailing a magnetic tape, but code changed hands from one site to another without hindrance. Indeed, we were careful to send the tapes home to a site under the care of a person. But, it was very much what we now call an "open source culture" as different groups modified the code. I never saw us fail to send a tape home to any licensed site.

As I said, this sharing was happening at a prodigious rate. Each site added features that were important to them or to fix issues that became acute in their environment. For instance, the original ARPANET code was done at University of Illinois (UofI); CMU would create disk recovery and Emacs, MIT a different Emacs and gave us compilers for many microprocessors; University of Delaware (UDEL) wrote a mailer, lots of people wrote editors, etc.

The most famous collection of modifications and additions to the UNIX "trade secrets" became the Berkeley Software Distribution (BSD) from the Electrical Engineering and Computer Science (EECS) department at UCB, distributed to their licensees (which of course all had an AT&T license). Since the 1960s and before the Computer Science part of the department has been created, the EE portion had operated its Industrial Liaison Office (ILO). One job of the ILO was to distribute "gratis" (free of any specific direct monetary charges) the sources to the code developed at UCB basically to anyone that asked, which in practice was the industrial partners around the world. Programs such a SPICE, SPLICE, MOTIS had seeded the electronics industry and mechanism to collect licenses for the codes and release tapes of them was already in place when the first BSD[16] for UNIX was made available as an update to the Sixth Edition. This was followed by 2BSD for the Seventh Edition/PDP-11 and then 3BSD and 4BSD for the VAX. Within a few years, the UC Berkeley team would get a DARPA contract and create the Computer System Research Group (CSRG) and

16 The first Berkeley UNIX Software Distribution was actually called BSD at the time of its release, but to distinguish it from later releases, modern people often refer to this as 1BSD.

start to push out BSD releases with prefixes of 4.1A, 4.1B, 4.1C, 4.2, 4.3, 4.4, 4.4Reno, 4.4Tahoe, and NET.

Here we see an interesting trend. By the time of the creation of UCB's CSRG team, we actually have an entire branch of the computing industry for UNIX, which has been named the "open systems" products to computer community at large (note the title on the SVR4 book I mentioned previously). We have small companies such as Masscomp and Sun or large ones such as Group Bull, Siemens, ICL, DEC, HP and IBM all producing products based on "open systems" technology. We even start to have a war within the community itself as to what is the definition of "UNIX".

Christensen Disruption Revisited

At the same time, many companies, including the original UNIX hardware manufacturer DEC, continued to produce competing computing systems that were not based on UNIX. Yet, UNIX which originally was not a product, is now an actual branch of the industry (the "Open Systems" branch) and it is thriving and growing at a tremendous rate. How did this happen?

As Christensen points out, disruptive technologies start off as a worse technology, but a different group of people values the technology. This is exactly how UNIX was created. It was targeted for a small inexpensive computer (PDP-7 originally, later PDP-11), written to provide a platform by programmers, for programmers, and as a place to experiment with ideas. It was not created originally as a large enterprise-wide computing service, not integrated together, nor delivered by a manufacturer. Even its humorous and irreverent name was a contrast to developers' experience with Multics (Organick, 1972).

As Christensen observes, in many ways that software would have been compared, looking at the Fourth, Fifth, Sixth or even the Seventh Editions of UNIX against its commercial peers of the mid-1970s (RT-11, RSX-11, RSTS, VAX/VMS), UNIX was considered to be "weaker or "not as good", if not a "toy" by many measures. Moreover, I mentioned at the beginning of this paper that programmers did not buy computers. But something extremely important had occurred: students like me had begun to graduate and were now in the workplace. We had been mentally contaminated with those open UNIX ideas, and while we may not have had the budget, we were asked by our bosses what to buy and we all wanted UNIX. What UNIX did for us was what we valued, not what the old systems had valued. And as Christensen predicts, the technology had gotten better — all while disrupting and displacing the mainstream because the growth curve of the UNIX community was much more rapid than the older curve.

Christensen predicts that worse technology will improve and over take the older technology, but what he did not predict and is interesting is that the case of UNIX, that because the technology was "open" / "free", its targeted users were the ones that could and did make it better! That is, this dynamic new market, which was made up of a whole new community of people who had previously been ignored in the old market, were the power that drove making the technology valuable. Many people in the community around the world worked to add to the core UNIX technology, because the core intellectual property had been freed by a side effect of that 1956 decree. Moreover, because it was in the open literature, published, taught, improved, the technology got strong because more people were able to contribute.

What does "Free" Mean

In perspective with the notion of "free software", was UNIX really free? The code was published. We read about it. We had access to it. Indeed, the UNIX source was licensed, and licensing did not change the freeness – as that is a separate quality. The ideas were published, the ideas were discussed. The source was shared within a community that had them for very little money. This is an important point: free does not mean it was without complete monetary cost (gratuity), but rather intellectually free (*libre*), which in this realm proved far more powerful. Some amount of money was spent for licensing. Some amount was spent on hardware. It is true that one can claim some people did not have access to the sources at all times (e.g. the 1980s and 1990s during the "UNIX Wars"), but the ideas were never locked away, they were never "closed."

So, the question is, at what point does an idea transition from a concept to thing? To me, while being able to use UNIX (the instance of it) made it real, and is what allowed us to come together, it was the ideas that really mattered and that was the part that was truly free.

In fact, the US courts came to the same conclusion, too. As I mentioned before, AT&T had considered UNIX a "trade secret." In the early 1990s, AT&T sued the UCB and a small firm made up of the former CSRG folks, called BSDi, on these grounds that they had used the trade secrets inappropriately and AT&T actually lost the case[17]. The finding of the court was simple: with the 1956 consent decree, AT&T was required to license its technology. When they licensed UNIX to UCB and similar academic institutions, then UCB and those institutions used to it teach people like me, we were by definition, "mentally contaminated" with AT&T's IP and they could not, by definition claim the IP was a trade secret

17 "USL *vs.* BSDi documents", by Dennis M. Ritchie, Bell labs website [URL: https://www.bell-labs.com/usr/dmr/www/bsdi/bsdisuit.html] (accessed 201709012004EDT).

any more. The ideas (the core IP) behind UNIX could not be locked up. The US courts declared the IP open and free because it had been published and described! (Pinheiro, 1987).

Economics *vs.* Technical Purity

The penultimate idea I want to explore about the driver for UNIX success is less obvious, but I think equally important. Engineering schools usually teach the best or optimal way to design different things. We take this thinking and often critique technical designs for some level of purity. Yet engineering is often made up of trade-offs, so one person's optimization might not be considered such by someone else.

When they started, Ken and Dennis were not trying to have an academically perfect system as they developed UNIX. They were not trying to build something glorious, or even a product for that manner. Technical purity was a non-goal; getting something they needed to get done was. Similarly, when users like me picked UNIX as a system, we picked it not because it was polished, came packaged, with a warranty or even included many programs traditionally associated with a computer OS (such as a real Fortran compiler); we picked it because it solved our problems inexpensively and effectively. UNIX was cost effective for us and we did not care about the deficiencies because its strengths were in features we cared about, which more than made up for its weakness. Because it was open, we could enhance it as we needed. And, enhance it we did. The key is that economics on UNIX was in our favor to enhance it. If you will, the cost to enter (purchasing the hardware was low), the software was basically free and student labor was inexpensive, so why not enhance it?

An International Sensation

One of the most wonderful parts of the UNIX story is that it crossed cultures and international boundaries. UNIX had been developed in the USA for programmers at AT&T. But the interesting thing is that the international language of programmers was problem solving and really is independent of most or many spoken languages. Even though UNIX lacked support to spoken languages other than English itself, most programmers spoke English already so that fact that it solved a set of problems that was globally common made a fine solution. It was not a single culture that made UNIX succeed, other than the one the technology created itself. As I hope I have demonstrated, UNIX was open and free so anyone, anywhere could be part of the community. As I pointed out, the best description of how the UNIX system worked was written in Australia, but it did not stop there. Professor Andrew Tanenbaum literally wrote volumes from

his offices in the Netherlands. It really was a worldwide effort with many hands being part.

In fact, in time, because UNIX was such an international sensation it made sense that it would become the leading system to support an internationalization effort, that was followed by many other systems later. For instance, the techniques used for the C programming language for systems like Microsoft's Windows were taken from the internationalization work that UNIX offered.

It is said imitation is the greatest form of flattery and UNIX started to be imitated. These "UNIX clones" sprang up all over the world. The first clone was done by an ex-AT&T person, Bill Plauger, who wrote a Sixth Edith clone called Idris[18]. I remember being extremely excited by a paper given by Michel Gien on the Sol Operating System at USENIX Summer 1983, as researchers in France were working on a clone of UNIX written in Pascal (Gien, 1983). The Pascal implementation would lead to the world-famous Chorus System done in C++. Tanenbaum and his students would write MINIX, a complete Seventh edition clone[19] and like Lions, published the full sources in his text book (Tanenbaum, 1987) as he wanted something with which he could teach, but at the time we were worried that he might be restricted. AT&T even investigated another Seventh Edition clone, called Coherent[20].

To me the height of this is today's Linux System, which clearly holds the UNIX banner high. The original kernel was written by a young Finn, we all know. Now the project has thousands of contributors around the world, with millions of lines of codes between the kernel and layered applications. What an amazing statement to the UNIX legacy, Ken, Dennis and the people that started it all.

The Virtuous Circle

Finally, you should notice a circular argument going on here, but that actually is the basis of my final point. UNIX was targeted at the academically-inclined, was economically accessible, and since the ideas which formed the basis of UNIX had been published in open literature and the implementation of those ideas had been made available to the academic community fundamentally without restriction, the "UNIX IP" was thus "free" and able to be openly examined, discussed, mani-

18 Idris was marketed by his then firm, Whitesmiths, Ltd. Traces of the code seem to be gone on the Internet, but you will reference to it.

19"MINIX 3", MINIX Group, Vrije Universiteit Amsterdam [URL: http://www.minix3.org] (accessed 201706191751EDT).

20 "Mark Williams Company Sources", Stephen A. Ness, Ness and MWC [URL: http://www.nesssoftware.com/home/mwc/source.php] (accessed 201706171421EDT). The Coherent story in interesting in itself. Dennis was one of the people that was asked to investigate it. AT&T eventually concluded it was unique enough as to not prosecute and the system stayed on the market, although was not financially successful.

pulated, even abused by the target users. Thus, my secondary thesis is that while its creators wrote UNIX for themselves, they freely shared it with the community and the sharing fed on the economics in a virtuous circle[21] that caused the community to grow. The more it grew, the more the demand for it became. As there is more demand for UNIX, there is more investment in its technology, and more people want to participate in developing the technology, which causes more demand – exactly what Krugman describes as a virtuous circle![22]

Bibliography

Bach M.J. (1986). *Design of the UNIX Operating System.* London, Prentice-Hall.

Christensen, C.M. (1997). *The Innovators Dilemma.* Boston, Harvard Business School Press.

Gien M. (1983). "The SOL Operating System", USENIX Association, 1983, *Proceedings of the Summer '83 USENIX Conference,* Toronto, Canada, July, 1983, pp.75-78.

Goodheart B. & Cox J. (1994). The Magic Garden Explained: The Internals of UNIX System V Release 4 An Open Systems Design, London, Prentice-Hall, ISBN 0-13-098138-9.

Leffler S.J., McKusick M.K., Karels M.J. & Quarterman J.S. (1989). *Design of and Implementation of the 4.3BSD UNIX Operating System,* Boston MA, Addison-Wesley Publishing Co.

Organick E. (1972). *The Multics System: An Examination of its Structure.* Cambridge (USA), MIT Press.

Pinheiro J. (1987). "AT&T Divestiture & the Telecommunications Market". *Berkeley Technical Law Journal,* 303, September 1987, Volume 2, Issue 2, Article 5.

Ritchie D.M. & Thompson K. (1974). "The UNIX Time-Sharing System". *Communications of the ACM,* July 1974, Volume 17, Number 7, pp.365-375.

Ritchie D.M. (1984). "The Evolution of the UNIX Time-Sharing System". *AT&T Bell Laboratories Technical Journal,* October 1984, Volume 63, Number 8, Part 2, pp.1577-1594.

Tanenbaum A.S. (1987). *Operating Systems: Design and Implementation.* London, Prentice-Hall.

21 See "The Fall and Rise of Development Economics", Paul Krugman [URL: http://web.mit.edu/krugman/www/dishpan.html] (accessed 201706191751EDT).

22 Thank you: This paper would not have been possible without a number of people encouraging me to write this history down for the future generations to read. In particular, my daughter Leah Cole, my sister Caroline Cole, my wife Margaret Marshall and the editorial team of *Cahiers d'histoire du Cnam* and their reviewers (especially Camille Paloque-Berges and a new friend from this endeavor, Loïc Petitgirard, who provided feedback and help in editing).

Enquête d'acteur

La conversion à Unix
Un exemple de prophétisme informatique ?

Laurent Bloch
Membre de l'Institut de l'Iconomie.

Unix survient une vingtaine d'années après l'invention de l'ordinateur, et une dizaine d'années après que quelques pionniers eurent compris qu'avec l'informatique une nouvelle science naissait, qu'ils eurent tenté de la faire reconnaître comme telle, et qu'ils eurent échoué dans cette tentative[1]. Certains traits d'Unix et certains facteurs de son succès procèdent de cet échec, et c'est de cette histoire qu'il va être question ici, selon la perception que j'en ai eue de ma position de praticien. Cette perception a été élaborée de façon rétrospective, aussi l'ordre chronologique n'est-il pas toujours respecté dans cet exposé. Ce qui suit est le récit de l'élaboration d'une vision personnelle, qui assume sa part de subjectivité.

Comme explicité par Benjamin Thierry, historien des techniques et de l'innovation à la Sorbonne, les pratiquants de disciplines en principe entièrement sous l'empire de la raison, telle l'informatique, ont fréquemment recours au vocabulaire religieux pour évoquer les aspects non rationnels de leur pratique, qui n'ont en principe pas droit de cité. L'histoire des sciences et des techniques ne nous laisse pas ignorer la présence en ces domaines de phénomènes de croyance qui peuvent aller jusqu'à des formes quasi-religieuses, et ainsi engendrer des manifes-

1 Ce texte est l'adaptation écrite d'une communication donnée par l'auteur, à la demande des organisateurs, lors du colloque « Unix en France et aux États-Unis : innovation, diffusion et appropriation » tenue au Cnam le 19 octobre 2017. Le sujet a été élaboré collectivement avec les organisateurs de la conférence et le discutant, Benjamin Thierry, et ce texte a été revu à la suite des discussions ayant eu lieu autour de l'allocution [URL : http://technique-societe.cnam.fr/colloque-international-unix-en-france-et-aux-etats-unis-innovation-diffusion-et-appropriation--945215.kjsp]. Ce texte est exceptionnellement placé sous une licence CC BY-NC-SA (Licence Creative Commons : Attribution + Pas d'utilisation commerciale + Partage dans les Mêmes Conditions).

tations typiques de la vie religieuse, telles que le prophétisme et la conversion. Le développement d'Unix m'a semblé doté d'une dimension prophétique, et pour développer ce point de vue j'emprunterai quelques idées au livre fondateur d'André Neher *L'essence du prophétisme* (1972). Pour rester fidèle (si j'ose dire) à l'esprit unixien, j'invite le lecteur à prendre cette dernière thèse *cum grano salis*[2].

Avant l'informatique

Entre 1936 et 1938 à Princeton, Alan Turing avait bien conscience de faire de la science, mais ne soupçonnait pas que ses travaux de logique seraient un jour considérés comme la fondation théorique d'une science qui n'existait pas encore, l'informatique (Bullynck, Daylight & De Mol, 2015 ; Haigh, 2014).

Dans les couloirs de l'IAS il croisait John von Neumann, parfois ils parlaient travail, mais pas du tout de questions de logique, domaine que von Neumann avait délibérément abandonné après la publication des travaux de Gödel (Corry, 2017). En fait, ils étaient tous les deux mathématiciens, et ils parlaient des zéros de la fonction ζ(s) et de l'hypothèse de Riemann (RH). Les efforts d'Alonzo Church pour éveiller l'intérêt de ses collègues pour les travaux sur les fondements de son disciple Turing (« On computable numbers, with an application to the Entscheidungsproblem », 1937) rencontraient peu de succès : la question apparaissait clairement démodée.

Ce n'est qu'à la rencontre de Herman Goldstine, et, par son entremise, des concepteurs de l'ENIAC Eckert et Mauchly, à l'été 1944, que von Neumann s'intéressa aux calculateurs, et sans le moins du monde établir un lien entre cet intérêt et les travaux de Turing dont il avait connaissance. Néanmoins, si Turing avait jeté les bases de l'informatique, von Neumann allait inventer l'ordinateur. Samuel Goyet a eu, lors d'une séance du séminaire Codes sources[3], une formule frappante et qui me semble exacte : avant von Neumann, programmer c'était tourner des boutons et brancher des fiches dans des tableaux de connexion, depuis von Neumann c'est écrire un texte ; cette révolution ouvrait la voie à la science informatique.

Lors d'une séance précédente du même séminaire, Liesbeth De Mol avait analysé les textes de von Neumann et d'Adele et Herman Goldstine[4], en montrant que tout en écrivant des programmes, ils n'avaient qu'une conscience encore imprécise du type d'activité à laquelle ils

2

3 Samuel Goyet, « Les interfaces de programmation (api) web : écriture informatique, industrie du texte et économie des passages », séminaire Codes source, séance du 8 juin 2017 [URL : https://codesource.hypotheses.org/241].

4 Liesbeth De Mol, « Un code source sans code ? le cas de l'Eniac », séminaire Codes source, séance du 22 juin 2016 [URL : https://codesource.hypotheses.org/219].

s'adonnaient. C'est donc une douzaine d'années après le « First Draft of a Report on the EDVAC »[5] de von Neumann (1945) que s'est éveillée la conscience de l'arrivée d'une science nouvelle, et j'en retiendrai comme manifestations les plus explicites la naissance du langage de programmation Algol, puis la naissance et la diffusion du système d'exploitation Multics. Il faudra encore une bonne quinzaine d'années pour que la bonne nouvelle se répande quelque peu parmi les praticiens de l'informatique, dont l'auteur de ces lignes, d'abord sous les espèces de la Programmation structurée (Dahl, Dijkstra & Hoare, 1972 ; Arsac, 1979), qui semait l'espoir d'une sortie du bricolage. Inutile de préciser que l'esprit de bricolage est encore présent parmi les praticiens.

Algol et Multics

Algol et Multics sont les cadavres dans le placard d'Unix, même si les meurtriers ne sont pas clairement identifiés.

Algol

La composition des comités de rédaction des rapports Algol successifs (Backus & al., 1960) et la teneur de leurs travaux[6] me semblent marquer un point de non-retour dans la constitution de l'informatique comme science. Jusqu'alors la programmation des ordinateurs était considérée un peu comme un bricolage empirique, qui empruntait sa démarche à d'autres domaines de connaissance

Fortran, le premier langage dit évolué, était conçu comme la transposition la plus conforme possible du formalisme mathématique. Du moins c'était son ambition, déçue comme il se doit[7], et le caractère effectuant du texte du programme, qui le distingue radicalement d'une formule mathématique, plutôt que d'être signalé, était soigneusement dissimulé, notamment par l'emploi du signe « = » pour désigner l'opération d'affectation d'une valeur à une variable, variable au sens informatique du terme, lui aussi distinct radicalement de son acception

5 John von Neumann, « First Draft of a Report on the EDVAC », Rapport technique, University of Pennsylvania, 1945 [URL : http://www.virtualtravelog.net/entries/2003-08-TheFirstDraft.pdf].

6 Voir le chapitre « The American Side of the Development of ALGOL », *in* Perlis (1981, pp. 75-91).

7 Un énoncé mathématique est essentiellement déclaratif : il décrit les propriétés d'une certaine entité, ou les relations entre certaines entités. Un programme informatique est essentiellement impératif (ou performatif) : il décrit comment faire certaines choses. Même écrit en style fonctionnel avec Lisp ou logique avec Prolog il sera finalement traduit en langage machine, impératif. Il est fondamentalement impossible de réduire l'un à l'autre, ou vice-versa, ils sont de natures différentes. Il est par contre possible, dans certains cas, d'établir une relation entre le texte d'un programme et un énoncé mathématique, c'est le rôle notamment des systèmes de preuve de programme. Ou, comme l'écrivent Harold Abelson et Gerald Jay Sussman (2011 [1985], p. 22) : « *In mathematics we are usually concerned with declarative (what is) descriptions, whereas in computer science we are usually concerned with imperative (how to) descriptions* ».

mathématique. La conception même du langage obscurcissait la signification de ses énoncés, ce que l'on peut pardonner à John Backus parce qu'il s'aventurait dans un domaine jamais exploré avant lui : « *Fortran montrait qu'un langage de programmation pouvait introduire de nouvelles abstractions qui seraient codées par un compilateur, plutôt que directement implémentées par le matériel* » (Foster, 2017). Cette affaire du signe « = » n'est pas si anecdotique qu'il y paraît, nous y reviendrons.

De même, Cobol se voulait le plus conforme possible au langage des comptables, et RPG cherchait à reproduire les habitudes professionnelles des mécanographes avec leurs tableaux de connexions.

Algol rompt brutalement avec ces compromis, il condense les idées de la révolution de la programmation annoncée par Alan Perlis (1981, p. 75) en tirant toutes les conséquences (telles que perçues à l'époque) de la mission assignée au texte d'un programme : effectuer un calcul conforme à l'idée formulée par un algorithme. La syntaxe du langage ne doit viser qu'à exprimer cette idée avec précision et clarté, la lisibilité du texte en est une qualité essentielle, pour ce faire il est composé de mots qui composent des phrases. L'opération d'affectation « := » est clairement distinguée du prédicat d'égalité « = ».

La composition du comité Algol 58 est significative : la plupart des membres sont des universitaires, l'influence des industriels est modérée, Américains et Européens sont à parité. En témoigne le tableau suivant, auquel j'ai ajouté quelques personnalités influentes qui n'appartenaient pas formellement au comité International Algebraic Language (IAL) initial, ou qui ont rejoint ultérieurement le comité Algol 60, ou qui simplement ont joué un rôle dans cette histoire.

Les années de naissance sont significatives : il s'agit de la génération 1925 (plus ou moins).

S'ils ne figurent pas au sein des comités, Edsger W. Dijkstra et Jacques Arsac ont contribué (avec Dahl, Hoare et beaucoup d'autres) à la systématisation de leurs idées sous la forme d'une doctrine, la programmation structurée (Dahl, Dijkstra & Hoare, 1972), qui a contribué à extraire ma génération de l'ignorance dans laquelle elle croupissait. Elle est souvent et abusivement réduite à une idée, le renoncement aux instructions de branchement explicite (GOTO), promulgué par un article célèbre de Dijkstra (1968), « Go to Statement Considered Harmful ». Il s'agit plus généralement d'appliquer à la programmation les préceptes du Discours de la Méthode, de découper les gros programmes compliqués en petits programmes plus simples[8], et ainsi d'éviter la programmation en « plat

8 Cette façon de diviser la difficulté a pris la forme du sous-programme, inventé par Maurice Wilkes, qui a permis la procédure et la fonction, puissants moyens de mise en facteur d'éléments de programmes réutilisables, et donc d'abstraction.

Friedrich L. Bauer	1924		Professeur Munich
Hermann Bottenbruch	1928	PhD Darmstadt	Ingénieur
Heinz Rutishauser	1918	PhD ETH Zürich	Professeur Zürich
Klaus Samelson	1918	PhD Munich	Professeur Munich
John Backus	1924	MS Columbia	Ingénieur IBM
Alan Perlis	1922	PhD MIT	Professeur Yale
Joseph Henry Wegstein	1922	MS U. Illinois	NIST
Adriaan van Wijngaarden	1916		U. Amsterdam
Peter Naur	1928	PhD Copenhague	Prof. Copenhague
Mike Woodger	1923	U. College London	NPL
Bernard Vauquois	1929	Doctorat Paris	Prof. U. Grenoble
Charles Katz	1927	MS U. Penn	Ingénieur Univac
Edsger W. Dijkstra	1930	PhD Amsterdam	U. of Texas, Austin
C. A. R. Hoare	1934		Prof. Oxford
Niklaus Wirth	1934	PhD Berkeley	Professeur Zürich
John McCarthy	1927	PhD Princeton	Professeur Stanford
Marcel-Paul Schützenberger	1920	PhD Math.	Professeur Poitiers
Jacques André	1938	PhD Math.	DR Inria
Jacques Arsac	1929	ENS	

de spaghettis », illisible et donc impossible à maintenir.

Rien n'exprime mieux l'aspiration de cette époque à la création d'une science nouvelle que le livre de Dijkstra *A Discipline of Programming* (1976) : l'effort pour exprimer de façon rigoureuse les idées les plus ardues de la programmation va de pair avec la recherche d'un formalisme qui ne doive rien aux notations mathématiques.

Pierre-Éric Mounier-Kuhn (2014), narre le succès initial rencontré dans les années 1960 en France par Algol 60, puis

son déclin au cours des années 1970, parallèle à celui connu en d'autres pays.

Algol 68 était un beau monument intellectuel, peu maniable pour la faible puissance des ordinateurs de l'époque, peu apte à séduire constructeurs d'ordinateurs et utilisateurs en entreprise avec des problèmes concrets à résoudre en temps fini.

Multics, un système intelligent

Multics est à la science des systèmes d'exploitation ce qu'est Algol à celle des langages de programmation : un échec public, mais la source d'idées révolutionnaires et toujours d'actualité qui sont à l'origine d'une science des systèmes d'exploitation. Certaines de ces idées ont d'ailleurs été largement empruntées par les créateurs d'Unix.

Multics est né en 1964 au MIT (Massachusetts Institute of Technology) dans le cadre d'un projet de recherche nommé MAC, sous la direction de Fernando Corbató. Nous ne reviendrons pas sur les circonstances bien connues dans lesquelles Ken Thompson et Dennis M. Ritchie ont créé Unix parce que le système Multics qu'ils utilisaient aux Bell Labs allait disparaître[9]. De Multics, ils voulaient conserver les caractères suivants :

- Le système est écrit non pas en assembleur, mais dans un langage de haut niveau (PL/1 pour Multics, C pour Unix). On a pu dire que le langage C était un assembleur portable.

- Le système de commandes qui permet de piloter le système est le même interpréteur qui permet à l'utilisateur d'exécuter des programmes et des commandes, et il donne accès à un langage de programmation. C'est le *shell*, inventé par Louis Pouzin pour CTSS, ancêtre de Multics.

- Le système de fichiers d'Unix doit beaucoup à Multics, d'où vient aussi l'idée d'exécuter chaque commande comme un processus distinct.

- Mais surtout, comme Dennis Ritchie l'a expliqué (1980), ce que lui et ses collègues des Bell Laboratories voulaient retrouver de Multics en créant Unix, c'était un système qui engendrait pour ainsi dire spontanément la communication et l'échange d'expériences entre ses adeptes.

Cette qualité, partagée par Multics et Unix, d'être propice à la création d'une communauté ouverte et communicative, mérite que l'on s'y arrête. Lorsque Multics a été introduit dans l'Institut statistique qui employait l'auteur de ces lignes à la fin des années 1970, il y a immédiatement cristallisé la formation

9 Cf. Laurent Bloch, « De Multics à Unix et au logiciel libre », Billet publié sur son site web en juillet 2014 [URL : https://laurentbloch.net/MySpip3/De-Multics-a-Unix-et-au-logiciel] et Les systèmes d'exploitation des ordinateurs – Histoire, fonctionnement, enjeux, livre en accès libre sur son site web [URL : https://laurentbloch.net/MySpip3/Systeme-et-reseau-histoire-et-technique].

d'une petite communauté intellectuelle, que la Direction n'a d'ailleurs eu de cesse de résorber parce qu'elle n'en comprenait pas la fécondité et qu'elle percevait son activité comme un gaspillage. D'innombrables expériences similaires ont eu lieu autour de Multics et d'Unix, sans qu'une cause unique puisse leur être attribuée. Le fait que ces systèmes aient été créés par des chercheurs, habitués à l'idée que la connaissance soit objet de partage gratuit et de communication désintéressée mais assortie de plaisir, est un élément. L'existence de logiciels commodes pour la création de textes, le courrier électronique et les forums en ligne a aussi joué, mais cette existence était-elle une cause ou une conséquence ? La nature programmable du *shell*, l'accès possible pour tous aux paramètres du système, inscrits dans des fichiers de texte ordinaires, encourageaient un usage intelligent du système, et l'intelligence va de pair avec l'échange.

De Multics, les créateurs d'Unix rejetaient la lourdeur, due en majeure partie au fait que les ordinateurs de l'époque n'étaient pas assez puissants pour le faire fonctionner confortablement : ils étaient trop chers, trop lents, trop encombrants.

Avenir de Multics

Il n'est paradoxal qu'en apparence de prédire l'avenir de ce système disparu. Certaines des idées les plus brillantes de Multics sont restées inabouties du fait des limites des ordinateurs de l'époque. Celle qui me tient le plus à cœur consiste à évacuer la notion inutile, voire nuisible, de fichier, pour ne garder qu'un seul dispositif d'enregistrement des données, la mémoire (mieux nommée en anglais *storage*), dont certains segments seraient pourvus de l'attribut de persistance[10].

Le fichier est un objet rétif à toute définition conceptuelle consistante, hérité de la mécanographie par les cartes perforées et la puissance d'IBM, mais sans aucune utilité pour un système d'exploitation d'ordinateur moderne. Il n'est que de constater la difficulté qu'il y a à expliquer à un utilisateur ordinaire pourquoi il doit sauvegarder ses documents en cours de création, et pourquoi ce qui est dans la mémoire est d'une nature différente de ce qui est sur le disque dur, terminologie qui révèle une solution de continuité injustifiée.

Les systèmes d'exploitation de demain, je l'espère, n'auront plus de fichiers, c'est déjà le cas de systèmes à micro-noyaux comme L4.

10 Cf. Laurent Bloch, « Mémoire virtuelle, persistance : faut-il des fichiers ? », Billet publié sur son site web en février 2014 [URL : https://laurentbloch.net/MySpip3/Memoire-virtuelle-persistance-faut].

Unix

Le crépuscule de Multics

Dennis Ritchie (1980) a décrit la période où les Bell Labs se retiraient du projet Multics. Le groupe de D. Ritchie, K. Thompson, M. D. McIlroy et Joseph F. Ossanna souhaitait conserver l'environnement de travail luxueux que Multics leur procurait à un coût d'autant plus exorbitant qu'ils en étaient les derniers utilisateurs. Pour ce faire ils allaient développer leur propre système sur un petit ordinateur bon marché et un peu inutilisé, récupéré dans un couloir : un PDP 7 de Digital Equipment. Unix était sinon né, du moins conçu.

Unix, un système en marge

Le livre de Peter H. Salus *A Quarter Century of UNIX* (1994) met en scène les principaux acteurs de la naissance d'Unix ; il est patent que c'est du travail « en perruque ». On est frappé en lisant ces aventures de découvrir que cette création, qui a eu des répercussions considérables dans les domaines scientifique et technique autant qu'industriel et économique, n'a vraiment été décidée ni par un groupe industriel, ni par un gouvernement, ni par aucun organisme doté de pouvoir et de moyens financiers importants. À la lecture du livre de Salus, quiconque a un peu fréquenté les milieux scientifiques d'une part, les milieux industriels de l'autre, ne peut manquer d'être frappé par le caractère décalé, pour ne pas dire franchement marginal, de la plupart des acteurs de la genèse unixienne.

Du côté de l'industrie informatique, la domination d'IBM était écrasante (95 % du marché mondial). Les constructeurs d'ordinateurs de gestion comme leurs clients ne se souciaient guère des aspects scientifiques de l'informatique, et là où l'informatique était appliquée à la science (essentiellement la physique) sa nature scientifique n'était pas mieux reconnue. Dans le monde universitaire l'informatique échouait à se faire reconnaître comme véritable objet intellectuel, et le système d'exploitation encore moins que les langages ou les algorithmes[11].

Les auteurs d'Unix

Or, que nous apprend Salus, auquel j'emprunte le générique avec lequel j'ai construit le tableau ci-dessous ? Thompson et Ritchie étaient chercheurs dans une entreprise industrielle. Au fur et à mesure de leur apparition, les noms de ceux qui ont fait Unix, parmi eux Kirk McKusick, Bill Joy, Eric Allman, Keith Bostic, sont toujours accompagnés d'un commentaire de la même veine : ils étaient étudiants *undergraduates* ou en cours de PhD, et soudain ils ont découvert qu'Unix était bien plus passionnant que leurs études. Bref, les auteurs

11 On trouvera une analyse approfondie de ces perceptions, pour le cas français, dans le livre de Pierre-Éric Mounier-Kuhn, *L'Informatique en France, de la Seconde Guerre mondiale au Plan Calcul* (2010, pp. 373-552).

Ken Thompson	1932	Berkeley, MS 1966	Ingénieur Bell Labs
Dennis Ritchie	1941	Harvard, PhD 1968	Ingénieur Bell Labs, Lucent
Brian Kernighan	1942	Princeton (PhD)	Ingénieur Bell Labs
Stephen Bourne	1944	PhD. Cambridge	Ingénieur Bell Labs etc.
Keith Bostic	1959		Ingénieur UC Berkeley
Joseph Ossanna	1928	BS Wayne State U.	Ingénieur Bell Labs
Douglas McIlroy	1932	Cornell, MIT PhD	Ingénieur Bell Labs
Kirk McKusick	1954	PhD Berkeley	Ingénieur, UC Berkeley
Eric Allman	1955	MS UC Berkeley	Ingénieur, UC Berkeley
Bill Joy	1954	MS UC Berkeley	Ingénieur Sun Microsystems
Özalp Babaoğlu	1955	PhD Berkeley	Professeur, U. Bologne
John Lions	1937	Doctorat Cambridge	Ing. Burroughs, U. Sydney
Robert Morris	1932	MS Harvard	Ingénieur Bell Labs, NSA
Mike Lesk		PhD. Harvard	Ingénieur Bell Labs
Mike Karels		BS U. Notre Dame	Ing. Berkeley

d'Unix n'ont jamais emprunté ni la voie qui mène les ingénieurs perspicaces vers les fauteuils de Directeurs Généraux, ni celle que prennent les bons étudiants vers la *tenure track*, les chaires prestigieuses, voire le Nobel[12].

Si l'on compare ce tableau avec celui des concepteurs d'Algol, on constate une différence de génération (1945 contre 1925), une plus faible propension à soutenir une thèse de doctorat, le choix de carrières d'ingénieur plutôt que d'universitaire, même si ces carrières se déroulent souvent dans un environnement de type universitaire. On notera que Joseph Ossanna avait aussi fait partie de l'équipe qui a réalisé Multics. On notera aussi la disparition des Européens, présents presque à parité dans le groupe Algol.

12 *Undergraduate* : BAC + 3 ou moins : PHD : thèse de doctorat ; *tenure track* : procédure de titularisation des enseignants-chercheurs aux États-Unis, plus longue et exigeante que son équivalent français.

Prophétisme informatique

Les circonstances évoquées ci-dessus et le caractère de leurs acteurs suggèrent un rapprochement avec les attitudes associées à un phénomène religieux, le prophétisme, tel qu'analysé par André Neher (1972). Il distingue (p. 47) quatre types de prophétisme :

- magique,
- social-revendicatif,
- mystique,
- eschatologique (c'est-à-dire apocalyptique, ou messianique).

Le prophétisme unixien relève assez clairement du type social-revendicatif, même si certains aspects de l'informatique peuvent plaider en faveur du type magique, et si certaines prophéties délirantes relatives à l'intelligence artificielle versent dans le style apocalyptique. Ken Thompson et Dennis Ritchie voulaient et annonçaient un style informatique approprié à leur style social et professionnel, et ils ont fait en sorte de le créer.

Un autre trait caractéristique du prophétisme est le scandale : un prophète qui ne créerait pas de scandale serait simplement ridicule. Neher explique (p. 255) que pour le prophète les temps ne sont pas accomplis, ils restent à construire, alors que les gens ordinaires préféreraient se reposer dans la léthargie du présent.

Les apôtres d'Unix n'ont pas manqué à leur mission sur le terrain du scandale. Ils ont bravé tant les informaticiens universitaires que le monde industriel, avec un succès considérable. Les puissances d'établissement (industriels, managers, autorités académiques du temps) ont nourri une véritable haine d'Unix, dont une des manifestations les plus ridicules fut la constitution d'un club des associations d'utilisateurs de systèmes d'exploitation mono-constructeurs, à l'intitulé spécialement formulé de sorte à écarter l'Association française des utilisateurs d'Unix et des systèmes ouverts, fondée en 1982, dont j'étais à l'époque membre du CA.

Unix occupe le terrain

Ces épisodes ont influencé tant la pratique informatique que le milieu social auxquels je participais, en général avec au moins une décennie de décalage. Cette expérience, renforcée par quelques décennies de discussions et de controverses avec quantité de collègues, m'a suggéré une hypothèse : au cours des années 1970, la génération d'Algol et de Multics a finalement perdu ses batailles, ses idées n'ont guère convaincu ni le monde industriel, dominé à l'époque par IBM, Digital Equipment et l'ascension des constructeurs japonais, ni le monde universitaire. La génération Unix a occupé le terrain laissé vacant, par une stratégie de guérilla (du faible au fort), avec de nouvelles préoccupations et de nouveaux objectifs. Pendant ce temps les géants de l'époque ne voyaient pas venir la vague micro-informatique qui allait profondément les remettre en cause.

La mission d'un universitaire consiste, entre autres, à faire avancer la connaissance en élucidant des problèmes compliqués par l'élaboration de théories et de concepts nouveaux. Les comités Algol et le groupe Multics ont parfaitement rempli cette mission en produisant des abstractions de nature à généraliser ce qui n'était auparavant que des collections d'innombrables recettes empiriques, redondantes et contradictoires. L'élégance d'Algol resplendit surtout dans sa clarté et sa simplicité.

La mission d'un ingénieur consiste en général à procurer des solutions efficaces à des problèmes opérationnels concrets. Nul ne peut contester que ce souci d'efficacité ait été au cœur des préoccupations des auteurs d'Unix, parfois un peu trop, pas tant d'ailleurs pour le système proprement dit que pour son langage d'implémentation, C.

Inélégances du langage C

Certains traits du langage C me sont restés inexplicables jusqu'à ce que je suive un cours d'assembleur VAX, descendant de leur ancêtre commun, l'assembleur PDP. J'ai compris alors d'où venaient ces modes d'adressage biscornus et ces syntaxes à coucher dehors, justifiées certes par la capacité exiguë des mémoires disponibles à l'époque, mais de nature à décourager l'apprenti. L'obscurité peut être un moyen de défense de techniciens soucieux de se mettre à l'abri des critiques.

Sans trop vouloir entrer dans la sempiternelle querelle des langages de programmation, je retiendrai deux défauts du langage C, dus clairement à un souci d'efficacité mal compris : l'emploi du signe « = » pour signifier l'opération d'affectation, et l'usage du caractère « NUL » pour marquer la fin d'une chaîne de caractères.

À l'époque où nous étions tous débutants, la distinction entre l'égalité et l'affectation fut une affaire importante. En venant de Fortran, les idées sur la question étaient pour le moins confuses, et il en résultait des erreurs cocasses ; après avoir fait de l'assistance aux utilisateurs pour leurs programmes Fortran, je parle d'expérience. L'arrivée d'une distinction syntaxique claire, avec Algol, Pascal, LSE ou Ada, sans parler de Lisp, permettait de remettre les choses en place : ce fut une avancée intellectuelle dans la voie d'une vraie réflexion sur les programmes.

Les auteurs de C en ont jugé autrement : ils notent l'affectation « = » et l'égalité « == », avec comme argument le fait qu'en C on écrit plus souvent des affectations que des égalités, et que cela économise des frappes. Ça c'est de l'ingénierie de haut niveau ! Dans un article du bulletin 1024 de la SIF, une jeune doctorante explique comment, lors d'une présentation de l'informatique à des collégiens à l'occasion de la fête de la Science, une collégienne lui a fait observer que l'expression « i = i+1 » qu'elle remarquait dans le texte d'un programme était fausse (Philippe, 2015). Cette collé-

gienne avait raison, et l'explication forcément controuvée qu'elle aura reçue l'aura peut-être écartée de l'informatique, parce qu'elle aura eu l'impression d'une escroquerie intellectuelle. Sa question montrait qu'elle écoutait et comprenait ce que lui disaient les professeurs, et là des idées durement acquises étaient balayées sans raison valable.

Pour la critique de l'usage du caractère « NUL » pour marquer la fin d'une chaîne de caractères, je peux m'appuyer sur un renfort solide, l'article de Poul-Henning Kamp « *The Most Expensive One-byte Mistake* » (Kamp, 2011). Rappelons que ce choix malencontreux (au lieu de représenter une chaîne comme un doublet « {longueur,adresse} ») contribue aux erreurs de débordement de zone mémoire, encore aujourd'hui la faille favorite des pirates informatiques. Poul-Henning Kamp énumère dans son article les coûts induits par ce choix : coût des piratages réussis, coût des mesures de sécurité pour s'en prémunir, coût de développement de compilateurs, coût des pertes de performance imputables aux mesures de sécurité supplémentaires…

Élégance d'Unix

Si le langage C ne manque pas de défauts, il est néanmoins possible d'écrire des programmes C élégants.

La première édition en français du manuel de système d'Andrew Tanenbaum (Tanenbaum & Bos, 2014) comportait le code source intégral de Minix, une version allégée d'Unix à usage pédagogique, qui devait servir d'inspiration initiale à Linus Torvalds pour Linux. Pour le commentaire du code source de Linux on se reportera avec profit au livre exhaustif et d'une grande clarté de Patrick Cegielski (2003) (la première édition reposait sur la version 0.01 du noyau, beaucoup plus sobre et facile d'accès que la version ultérieure commentée pour la seconde édition à la demande de l'éditeur).

On trouvera à la fin de cet article un exemple de code source extrait du *scheduler* (éventuellement traduit par ordonnanceur) de Linux 0.01. Le *scheduler* est l'élément principal du système d'exploitation, il distribue le temps de processeur aux différents processus en concurrence pour pouvoir s'exécuter.

Les codes de ces systèmes sont aujourd'hui facilement disponibles en ligne[13]. Par souci d'équité (d'œcuménisme ?) entre les obédiences on n'aura garde d'omettre BSD, ici la version historique 4BSD[14]. Quelques extraits figurent à la fin du présent texte.

13 Par exemple : « Minix 3.2.1, scheduler main routine », publié par Andrew Tanenbaum et Nick Cook [URL : http://homepages.cs.ncl.ac.uk/nick.cook/csc2025/minix/3.2.1/usr/src/servers/sched/main.c] ; « Linux v0.01 scheduler », publié par Linus Torvalds [URL : https://kernel.googlesource.com/pub/scm/linux/kernel/git/nico/archive/+/v0.01/kernel/sched.c].

14 « Freebsd 11.1 scheduler », publié par Jeffrey Roberson et Jake Burkholder, 2002-2007 [URL : https://github.com/freebsd/freebsd/blob/master/sys/kern/sched_ule.c].

L'élégance d'Unix réside dans la sobriété et la simplicité des solutions retenues, qui n'ont pas toujours très bien résisté à la nécessité de les adapter à des architectures matérielles et logicielles de plus en plus complexes. Ainsi, la totalité des 500 super-ordinateurs les plus puissants du monde fonctionnent sous Linux, ce qui implique la capacité de coordonner plusieurs milliers de processeurs – 40 460 pour le chinois Sunway TaihuLight qui tenait la corde en 2016, dont chacun héberge plusieurs centaines de processus : le *scheduler* ultra-concis du noyau Linux v0.01 dont on trouvera le texte ci-dessous en serait bien incapable.

Autre élégance des versions libres d'Unix (Linux, FreeBSD, NetBSD, OpenBSD…) : le texte en est disponible, et les paramètres variables du système sont écrits dans des fichiers de texte lisible, ce qui permet à tout un chacun de les lire et d'essayer de les comprendre.

Conclusion

Finalement, les universitaires orphelins d'Algol et de Multics se sont convertis en masse à Unix, ce qui assurait son hégémonie sans partage. La distribution de licences à un coût symbolique pour les institutions universitaires par les Bell Labs, ainsi que la disponibilité de compilateurs C gratuits, furent pour beaucoup dans ce ralliement, à une époque (1982) où le compilateur Ada que j'avais acheté à Digital Equipment, avec les 80 % de réduction pour organisme de recherche, coûtait 500 000 francs.

Unix a permis pendant un temps la constitution d'une vraie communauté informatique entre universitaires et ingénieurs, ce qui fut positif. Mon avis est moins positif en ce qui concerne la diffusion du langage C : pour écrire du C en comprenant ce que l'on fait, il faut savoir beaucoup de choses sur le système d'exploitation et l'architecture des ordinateurs, savoirs qui ne peuvent s'acquérir que par l'expérience de la programmation. C n'est donc pas un langage pour débutants.

Si le langage C est relativement bien adapté à l'écriture de logiciel de bas niveau, typiquement le système d'exploitation, je reste convaincu que son usage est une torture très contre-productive pour les biologistes (ce sont ceux que je connais le mieux) et autres profanes obligés de se battre avec malloc, sizeof, typedef, struct[15] et autres pointeurs dont ils sont hors d'état de comprendre la nature et la signification. La conversion à C n'a pas vraiment amélioré la pratique du calcul scientifique ; la mode récente de Python est alors un bienfait parce qu'au moins ils pourront comprendre (peut-être) le sens des programmes qu'ils écrivent.

15 Commandes du système Unix.

Annexe : code source

```
/*
 *  'schedule()' is the scheduler function. This is GOOD CODE! There
 * probably won't be any reason to change this, as it should work well
 * in all circumstances (ie gives IO-bound processes good response etc).
 * The one thing you might take a look at is the signal-handler code here.
 *
 *   NOTE!!  Task 0 is the 'idle' task, which gets called when no other
 * tasks can run. It can not be killed, and it cannot sleep. The 'state'
 * information in task[0] is never used.
 */
void schedule(void)
{
       int i,next,c;
       struct task_struct ** p;
/* check alarm, wake up any interruptible tasks that have got a signal */
       for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
              if (*p) {
                     if ((*p)->alarm && (*p)->alarm < jiffies) {
                                   (*p)->signal |= (1<<(SIGALRM-1));
                                   (*p)->alarm = 0;
                            }
                     if ((*p)->signal && (*p)->state==TASK_INTERRUPTIBLE)
                            (*p)->state=TASK_RUNNING;
              }
/* this is the scheduler proper: */
       while (1) {
              c = -1;
              next = 0;
              i = NR_TASKS;
              p = &task[NR_TASKS];
              while (--i) {
                     if (!*--p)
                            continue;
                     if ((*p)->state == TASK_RUNNING && (*p)->counter > c)
                            c = (*p)->counter, next = i;
              }
              if (c) break;
              for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
                     if (*p)
                            (*p)->counter = ((*p)->counter >> 1) + (*p)->priority;
       }
       switch_to(next);
}
```

Scheduler du noyau Linux v0.01

Commentaire sur cette annexe

Avec l'aide de Patrick Cegielski (2003, p. 196) on observe que la variable c représente la priorité dynamique de la tâche considérée[16]. Le *scheduler* parcourt la table des tâches, et parmi celles qui sont dans l'état « TASK_RUNNING », c'est-à-dire disponibles pour s'exécuter, il sélectionne celle qui a la priorité dynamique la plus élevée et lui donne la main : « switch_to(next) ; ».

Bien que pour un texte en langage C ce *snippet* (fragment de code) soit relativement propre, il illustre la raison de mes réticences à l'égard de ce langage. Et lorsque Peter Salus (1994, p. 77) écrit que C est un langage différent des autres langages de programmation, plus proche d'un langage humain comme l'anglais, on peut se demander s'il a un jour écrit une ligne de programme, en C ou autre chose.

16 Dans le contexte de Linux v0.01 tâche est synonyme de processus. Ultérieurement il peut y avoir une certaine confusion entre processus, tâche et *thread*, mais il s'agit toujours d'un programme candidat à l'exécution auquel le *scheduler* doit décider de donner ou non la main. La valeur de la variable *jiffies* est le nombre de tops d'horloge depuis le démarrage du système.

Bibliographie

Abelson H., Sussman G. J. & Sussman J. (2011 [1985]). *Structure and Interpretation of Computer Programs.* Cambridge, Massachusetts : MIT Press.

Arsac J. J. (1979). « Syntactic source to source transforms and program manipulation ». *Communications of the ACM,* vol. 22, n° 1, pp. 43-54.

Backus J.W., Bauer F. L., Green J., Katz C., McCarthy J., Perlis A. J., Rutishauser H., Samelson K., Vauquois B., Wegstein J. H., van Wijngaarden A. & Woodger M. (1960). « Report on the algorithmic language algol 60 ». *Communications of the ACM,* vol. 3, n° 5, pp. 299-314.

Bullynck M., Daylight E. G. et De Mol L. (2015). « Why did computer science make a hero out of Turing ? ». *Communications of the ACM,* vol. 58, n° 3, pp. 37-39.

Cegielski P. (2003). *Conception de systèmes d'exploitation – Le cas Linux.* Paris : Eyrolles.

Corry L. (2017). « Turing's pre-war analog computers: The fatherhood of the modern computer revisited ». *Communications of the ACM,* vol. 60, n° 8, pp. 50-58.

Dahl O.J., Dijkstra E. W. & Hoare C. A. R. (éds) (1972). *Structured Programming.* London, UK : Academic Press Ltd.

Dijkstra E. W. (1968). « Letters to the editor: Go to statement considered harmful ». *Communications of the ACM,* vol. 11, n° 3, pp. 147-148.

Dijkstra E. W. (1976). *A Discipline of Programming.* Englewood Cliffs, NJ : Prentice-Hall PTR, 1st edition.

Foster J. S. (2017). « Shedding new light

on an old language debate ». *Communications of the ACM,* vol. 60, n° 10.

Haigh T. (2014). « Actually, Turing did not invent the computer ». *Communications of the ACM,* vol. 57, n° 1, pp. 36-41.

Kamp P.-H. (2011). « The most expensive one-byte mistake ». *ACM Queue* 9/7 [URL : http://queue.acm.org/detail.cfm?id=2010365].

Neher A. (1972). *L'essence du prophétisme*. Paris : Calmann-Lévy.

Perlis A. J. (1981). *History of programming languages*. New York, NY : ACM, 1981.

Philippe A.-C. (2015). « Quand une doctorante fait des heures supplémentaires ». *Bulletin de la société informatique de France* 1024, Hors-série n° 1, [URL : http://www.societe-informatique-de-france.fr/wp-content/uploads/2015/03/1024-hs1-philippe.pdf].

Ritchie D. M. (1980). « The Evolution of the Unix Time-sharing System ». *Lecture Notes in Computer Science*, vol. 79 (« Language Design and Programming Methodology »).

Salus P. H. (1994). *A Quarter Century of UNIX*. Reading, Massachusetts : Addison-Wesley.

Tanenbaum A. S. et Bos H. (2014). *Modern Operating Systems*. Upper Saddle River, NJ : Pearson.

Mounier-Kuhn P.-É (2010). *L'Informatique en France, de la Seconde Guerre mondiale au Plan Calcul. L'émergence d'une science*. Paris : Presses de l'Université Paris-Sorbonne.

Mounier-Kuhn P.-É (2014). « Algol in France : From universal project to embedded culture ». *IEEE Annals of the History of Computing,* vol. 36, n° 4.

Turing, A.M. (1937). « On computable numbers, with an application to the Entscheidungsproblem ». *Proceedings of the London mathematical society,* 2(1), pp. 230-265.

Témoignage

UNIX vu de province : 1982–1992

Jacques Talbot
ex-Bull Grenoble.

Résumé

Au sein du groupe Bull, une poignée d'informaticiens en province (sur le site grenoblois du groupe) ont tenté d'orienter la politique technique de la compagnie dans les années 1982-1992, autour du système d'exploitation UNIX. Ce texte décrit le contexte dans lequel s'est développée cette aventure, et les épisodes clés entre les deux évènements majeurs dans cette histoire : en 1982, Bull acquiert la licence de l'ordinateur SM90 qui fonctionne sous UNIX et en 1992, l'accord entre Bull et IBM autour d'AIX (système d'exploitation de type UNIX développé par IBM) qui ouvre une nouvelle ère. Ce texte montre comment le groupe est pris entre les intérêts du site en local et la politique globale de Bull, et il relate sa participation à la dynamique de développement international d'UNIX.

Mots-clés : UNIX, Bull, Systèmes d'exploitation, Systèmes ouverts, Standardisation d'UNIX.

Introduction

Ce texte est un témoignage personnel[1] sur l'histoire d'UNIX dans Bull, vue de Grenoble : arrivé en 1980 à la SEMS Grenoble (qui intègre le groupe Bull en 1981), j'ai participé à la définition du noyau UNIX chez Bull, en prenant la fonction de chef de projet d'une machine UNIX de 1987 à 1990, puis fonction d'architecte de l'infrastructure logicielle.

La vue de province introduit nécessairement un biais, mais il est de même pour tout regard, qu'il soit national, international ou local. Le chef d'œuvre littéraire *Les enfants de Sanchez* de l'anthropologue états-unien Oscar Lewis,

1 Ce n'est pas un travail d'historien à proprement parler, qui aurait nécessité de rassembler beaucoup de sources et d'interviewer les acteurs de l'époque. Il repose avant tout sur mes souvenirs et des cahiers de travail épars.

qui relate la vie d'une famille mexicaine relatée par plusieurs de ses membres, montre que les points de vue de chacun sur un même événement peuvent paraître remarquablement divergents. Il en est de même ici.

Ce texte traduit le point de vue de la poignée de techniciens grenoblois qui ont tenté d'orienter la politique technique de la compagnie dans ces années, dans l'intérêt de leur établissement d'Échirolles[2] et tel qu'ils le comprenaient à l'époque. Le texte est restreint à la période 1982-1992 correspondant à deux événements importants. En 1982, Bull acquiert à la sauvette la licence de la SM90 qui fonctionne sous UNIX. En 1992, l'accord entre Bull et IBM autour d'AIX (système d'exploitation de type UNIX développé par IBM) a ouvert une nouvelle ère[3]. Le texte indique dans quel contexte s'est déroulée cette aventure, qui sera relatée à travers quelques épisodes clefs, dont le lecteur pourra essayer de tirer une morale aujourd'hui. Je me suis volontairement limité aux aspects de l'histoire que je connais le mieux, négligeant des éléments (comme le SPS 9, TRIX[4], l'impact des SGBDR…) que j'ai observés de plus loin[5].

Le monde de l'informatique autour de 1982

Pour comprendre les choix faits à Bull, il faut avoir à l'esprit les débats qui agitent le monde de l'informatique, à tous les échelons : international, européen, français et grenoblois.

Le contexte international dans l'industrie informatique en 1982

En ce début des années 1980, l'informatique est structurée par l'opposition entre les grands systèmes (*mainframe*) et les mini-ordinateurs. L'ordinateur personnel n'a pas encore vraiment émergé comme phénomène, que ce soit chez les particuliers ou dans les entreprises, puisque le PC d'IBM est sorti en 1981 et le Mac d'Apple ne sortira qu'en 1984.

Le *mainframe* est un gros ordinateur, cher, souvent loué et destiné à l'informatique de gestion ou, marginalement, scientifique. Le leader incontesté (encore aujourd'hui d'ailleurs) est IBM, imité et suivi aux États-Unis par les survivants des « sept nains » (les constructeurs du BUNCH : Burroughs, Univac, NCR, Control Data et Honeywell) et en Europe par cinq champions nationaux : Bull en France, ICL au Royaume-Uni, Siemens et Nixdorf en Allemagne et Olivetti en Italie.

2 « Bull Grenoble » est en fait localisé à Échirolles, une banlieue au sud de la ville.

3 Sur ces points, voir l'encadré de Ph. Picard et M. Elie dans ce volume.

4 Voir l'article de René Chevance, « Contribution à l'histoire d'UNIX chez Bull », publié en 2004 sur le site de la Fédération des Équipes Bull [URL : www.feb-patrimoine.com/projet/unix/histoire_unix_chez_bull.doc].

5 Remerciements : je remercie Denis Bram, Jacques Febvre, Jean Papadopoulo et Jez Wain qui ont bien voulu relire le brouillon et me fournir des éléments oubliés ou ignorés.

Le système d'exploitation (*Operating system* – OS) de référence est donc IBM MVS (Multiple Virtual Storage), qui a changé plusieurs fois de nom depuis (OS/390, z/OS), caractérisé par sa sophistication, sa complexité et ses modalités *batch*, temps partagé et transactionnel.

Le mini-ordinateur est plus petit, moins cher, en général acheté et destiné aux marchés temps réel, industriel, scientifique. Le leader est Digital Equipment, numéro 2 de l'informatique de l'époque, aujourd'hui disparu. Digital a de nombreux concurrents, car développer un mini-ordinateur et son logiciel nécessite un investissement bien moins important que pour les *mainframe*, mais tous les concurrents sont beaucoup moins puissants que lui sur le marché. Son modèle emblématique est le VAX 11/780, une machine 32-bit qui a permis en 1978 de sortir la mini-informatique de l'enfermement dans les 64 Ko d'une architecture 16-bit. L'OS VAX/VMS est un exemple d'élégance et de simplicité efficace.

Une autre opposition est très importante : l'informatique et les télécoms constituent deux silos qui se parlent peu, n'ont pas la même culture technique et commerciale et se méfient l'un de l'autre, car la frontière sur le marché est mal délimitée. Les réseaux informatiques, qui ont besoin bien sûr de liens de communication, sont très peu développés, compliqués et chers.

Durant la décennie 1980, des débats commencent à apparaître dans la communauté académique et industrielle sur la meilleure façon de distribuer présentation, métier et données (les « tier » du modèle 3-tier) entre les différentes machines, PC, UNIX, *mainframe*. On parle alors d'informatique distribuée pour opposer les *mainframe* et les autres. Ce terme est assez ambigu, car il y a des dizaines de manières de distribuer un traitement informatique. Le modèle de distribution actuellement dominant sur le Web, *smartphone* et services web en est un à très gros traits, contraint par les propriétés de performance et de fiabilité du réseau Internet. Les rêves d'OS distribué sont passés à la trappe de l'histoire.

UNIX, qui a été inventé aux Bell Labs (le laboratoire de recherche de l'opérateur télécom AT&T !) et en est sorti en 1975 (première licence source), est un objet technique sans importance sur le marché en 1982. Rappelons qu'UNIX est le père de Linux, de MacOS, d'iOS sur iPhone, d'Android sur la plupart de smartphones, de tous les OS des gros serveurs (AIX, HP-UX) et constitue donc une des matrices (avec Windows et MVS) de l'informatique actuelle.

Une caractéristique essentielle de l'ensemble de l'informatique mondiale d'alors est son aspect dit « propriétaire ». Chaque constructeur développe la totalité de la pile technologique : processeur, périphériques, OS, protocoles de réseau, applications. Cette démarche est délibérée pour contrôler le client. Celui-ci est donc enfermé à vie avec son fournisseur. Cette frustration génère de la part des

clients une demande pour des « systèmes ouverts », même si on ne sait pas bien encore ce que c'est.

Sun Microsystems, qui sera la star UNIX jusqu'en 2001 (au moment ou éclate la bulle Internet), est fondé en 1982. L'un des architectes d'UNIX BSD, Bill Joy, en est le patron logiciel. Ce sera la seule grande société dont UNIX a été le porte-drapeau en matière d'OS.

Années 1970 : l'échec des projets européens en informatique

À cette époque comme aujourd'hui, l'Europe est à la recherche de son unité, en informatique et ailleurs. Or, l'informatique européenne se remet mal de l'échec du projet Unidata dans les années 1973-1975. Cet échec introduira une telle méfiance entre les acteurs que tout projet ultérieur d'unification face aux Américains échouera, contribuant à la disparition *de facto* de l'informatique européenne. Le rôle et les motivations des politiques et des industriels français dans l'échec d'Unidata ont été abondamment documentés. Ce fantôme est le frère maudit d'Airbus et de l'Agence spatiale européenne (ESA).

L'industrie française de l'informatique ré-organisée en 1981

À son arrivée au pouvoir en 1981, la gauche a nationalisé et fusionné sous le nom de Bull, le fabricant français de *mainframe*, CII Honeywell-Bull et les deux constructeurs principaux de mini-informatique, la SEMS et Transac, sous la forme de filiales. Le P.-D.G. sera Jacques Stern jusqu'en 1989, assisté de Francis Lorentz, qui lui succédera de 1989 à 1992. Ce duo n'est pas sans talent, mais sa gestion d'UNIX a été plutôt calamiteuse.

Bull a 26 000 employés en 1983 et culminera à 43 000 en 1988. De nos jours, les effectifs de Bull sont dilués dans ATOS et ça n'a plus guère de sens de les compter.

CII Honeywell-Bull est le fruit de la fusion en 1975 de la CII et de Honeywell-Bull, lui-même issu de la fusion en 1970 de Bull et de la branche informatique du constructeur américain Honeywell.

SEMS est le fruit de la fusion en 1976 entre la branche informatique de la Télémécanique et la division mini-ordinateurs de la CII. La Télémécanique, localisée en région grenobloise, a conçu le mini-ordinateur 16-bit Solar pour concurrencer le PDP-11 de DEC. Sa panoplie d'OS ressemble à celle de DEC et est marquée par une certaine simplicité. Un avatar du Solar assure encore en 2017 le contrôle-commande et donc la sécurité de la plus grande partie de nos 58 réacteurs nucléaires. La ligne mini-ordinateur 16-bit apportée par la CII est le Mitra. Il a été conçu pour les marchés télécom, militaire et industriel ; son OS, MMT2 est un mini OS de mainframe, avec toute la complexité associée. La direction tech-

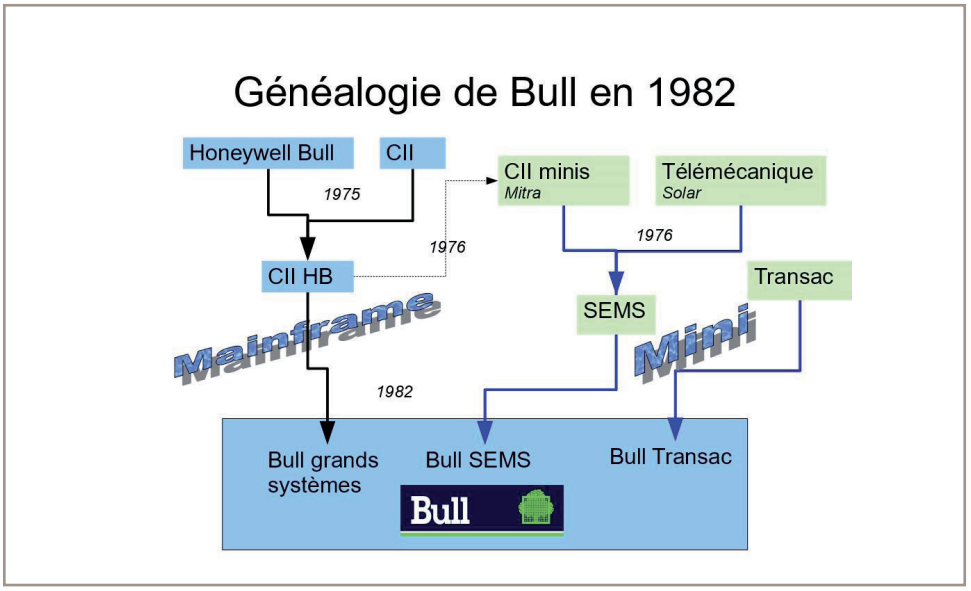

Figure 1 – Généalogie de Bull en 1982

nique SEMS est regroupée à Echirolles, en banlieue grenobloise. La partie Mitra a été mutée là en 1980/81.

La problématique mondiale vue de Grenoble

La « cellule architecture » (au sein de Bull SEMS) à Échirolles est petite. On peut dire qu'elle se limite à Jacques Febvre, mon chef à l'époque, brillant et sympathique, et moi-même. Notre culture technique est résolument celle de mini-informaticiens. Dans l'esprit du temps, en matière d'architecture, nous croyons à une certaine simplicité, avec en particulier des espaces virtuels grands et plats, à l'opposé de la segmentation qui a été l'enfer du Solar et surtout du Mitra. Même si nous regardons avec gourmandise les approches de type machine objet, comme l'IBM System/38 (sorti en 1979, de nos jours émulé sur une machine Power), les adressages à capacité de Multics et les projets de recherche comme le SCQM (680 000) de Jacques Leroudier, nous n'y croyons guère pour nous-mêmes. C'est d'ailleurs Jacques Leroudier qui nous a initiés à UNIX en 1980. Qu'il en soit remercié !

Il s'agit de promouvoir des développements qui soient à la portée d'une équipe grenobloise relativement modeste. *Volem viure al pais*[6] reste la motivation numéro un. Il faut donc d'abord sortir des

6 « *Nous voulons vivre au pays* ». C'est dans les années 1970 le slogan des paysans du Larzac.

64 Ko imposés par les adressages 16-bit, et nous sommes largement en retard sur DEC et Data General. En matière de langage, nous voulons bien sûr oublier l'assembleur et autres PL/16 pour écrire les OS.

Nous croyons aussi qu'il n'est plus possible d'enfermer le client dans un filet propriétaire (mais cette conviction n'est guère partagée par le management). La raison principale pour ce faire est le problème du catalogue d'applications. Comment attirer vers sa plateforme les industriels du logiciel (ISVs, Independent Software Vendor) qui émergent partout ? Il faut au minimum avoir une normalisation des APIs (l'interface programmatique, Application Programming Interface), pour que l'ISV ait un code source unique et un portage facile, et, mieux, avoir en plus un ABI (Application Binary Interface) pour qu'un binaire unique puisse fonctionner sur des OS divers. On verra que l'API normalisé aura du succès, mais l'ABI restera un Graal inatteignable. De nos jours, cette problématique semble bizarre puisqu'on a un catalogue d'application Google Play Store : quel est le problème ? Cela n'est pas problématique car il y a dessous un OS unique, Android.

UNIX en 1982

UNIX V7, publié en 1979 par les Bell Labs, est la première version réellement portable, même si sa cible est le DEC PDP/11. Elle est écrite en langage C, et documentée dans un *listing* source fameux, le Lions[7], qui frappe par sa petitesse (10 000 lignes de code). Elle est, encore pour quelque temps, libre de droits, avant qu'AT&T ne réalise qu'on peut gagner de l'argent avec les licences UNIX. C'est la version qui va permettre à Sun de se lancer, et celle qui sera portée sur Intel sous le nom Xenix. UNIX a puisé des concepts dans Multics, mais vise la simplification, peu d'abstractions et une certaine élégance.

En parallèle, une équipe à l'université de Berkeley développe une version dite BSD. C'est la première à fonctionner en mode 32-bit sur le VAX et à gérer la pile TCP/IP (en 1983, avec 4.2 BSD). En France, dans la sphère universitaire, UNIX est à la mode, mais pour être indépendant de la licence AT&T, il faut le réécrire et c'est le langage Pascal qui est choisi. C'est le projet SOL de Michel Gien à l'INRIA, vers 1980. Comme exercice intellectuel, c'est intéressant, mais il ne nous semble guère raisonnable de *forker*[8] ainsi un UNIX français alors qu'UNIX est loin de sa maturité. On peut objecter bien sûr que c'est proche de ce qu'a fait Linus Torvald en créant Linux avec le succès que l'on sait, mais c'était en 1992 avec un UNIX stabilisé qui n'évoluait plus guère.

7 Le Lions est en fait la V6. (N.D.E. : Un *listing* est une impression de lignes de code ou de données pour vérification et commentaires très utilisée à l'époque quand les écrans interactifs n'étaient pas encore généralisés).

8 N.D.E. : un *fork* est une version dérivée du code source.

Dans les rencontres UNIX de l'époque, le gag favori était de rester coi devant un transparent rempli de *pipes*, *awk*, *grep*, *sed*[9] et de lignes de *shell* ou de C cabalistiques à souhait. Puis on enlevait sa cravate en s'écriant « *I forgot you can't understand this with a tie on !* ». Ce n'est pas aussi anecdotique qu'il y paraît. UNIX s'est épanoui en tant que phénomène générationnel contre la culture *mainframe* symbolisée par la cravate de l'employé IBM modèle.

Le contexte de la guerre des réseaux

La synergie entre UNIX et TCP/IP est un point capital pour l'histoire. Dans les années 1980 se livre une guerre des piles protocolaires qui oppose un modèle normalisé à l'ISO et dénommé OSI et TCP/IP, développé à l'origine dans la DARPA[10], puis géré par l'IETF[11]. Il s'agit de surmonter l'absurde prolifération des piles propriétaires, chaque constructeur ayant la sienne : l'industrie globalement dépense donc des fortunes pour faire de trop nombreuses passerelles plus ou moins dysfonctionnelles.

La pile ISO/OSI est chère au cœur des Français de Bull, car beaucoup de ses concepts sont issus des travaux de l'équipe menée par Louis Pouzin et Hubert Zimmermann à l'INRIA, du CNET[12] et de l'architecture DSA de Bull. Après moult tergiversations, les Européens se sont mis d'accord en mode « comité » (*design by committee*) sur ce modèle et ses sept couches, et l'ont vendu à l'ISO. Le mode « comité » a l'inconvénient de générer des protocoles très complets, trop compliqués et mal testés.

À l'opposé, la démarche TCP/IP issue d'Arpanet privilégie, selon le slogan inventé a posteriori en 1992, un « *consensus approximatif et du code qui marche*[13] ». C'est moins ambitieux, mais plus simple. On sait que cette pile conquerra le monde avec Internet, mais en 1982, la messe n'est pas dite.

Ainsi, avec le portage de TCP/IP dans UNIX par Berkeley en 1983, un duo très puissant voit le jour.

9 N.D.E. : commandes du système UNIX.

10 La DARPA est la « Defense Advanced Research Project Agency », une agence du département de la défense états-unien. (N.D.E. : ce développement a eu lieu au sein du projet ARPANET, considéré comme le prototype d'Internet.)

11 L'IETF (Internet Engineering Task Force) est un groupe relativement informel qui gère les standards Internet depuis 1986.

12 Le CNET était le laboratoire de recherche français en télécommunications. Son descendant actuel est Orange Labs.

13 « *Rough consensus and running code* », est le slogan de l'IETF.

Retour sur terre : UNIX à Bull

Cette deuxième partie décrit quelques moments clés du point de vue grenoblois, jalonnant l'histoire d'UNIX à Bull durant cette décennie.

Une fois refermée cette analyse du contexte stratégique de la mini-informatique, revenons à Échirolles. Seules quelques personnes se préoccupent de ces problèmes à moyen terme. Le stratège en chef de la SEMS, Jean Helein, est toujours basé à Louveciennes. Assez âgé, très prudent, s'il a une vision, il ne nous en fait pas part. Nous allons donc errer quelque temps dans différentes directions avant d'emprunter résolument le chemin UNIX.

« Briques de base » et « nouvelle gamme Bull »

Dans l'euphorie de la prise de pouvoir, la gauche a lancé en 1981 un chantier national, les « Briques de base de l'informatique française ». Il s'agit de réunir industriels et universitaires pour définir des objets matériels et logiciels qui pourraient servir à un LEGO informatique. C'est un épisode bien oublié, introuvable sur le Web[14]. Le management des industriels n'y croit guère ; néanmoins, pour grappiller quelques subventions, Jacques Febvre et moi-même nous retrouvons à discuter ces objets avec quelques chercheurs et industriels. Le fait que la mission soit confiée à deux provinciaux est d'ailleurs symptomatique de sa faible crédibilité. Nous nous apercevons rapidement qu'on ne va nulle part, par manque de clarté des objectifs, et surtout manque de leader crédible. L'industrie et la recherche ont du mal à travailler ensemble, les méfiances réciproques, souvent injustifiées, sont nombreuses. Cette initiative sera rapidement close par suppression des crédits publics, sans avoir rien donné. Nous en tirerons la leçon qu'une certaine modestie des ambitions est de mise.

Dès novembre 1982, on nous demande de laisser de côté l'aspect « briques de base » et de nous concentrer sur la « nouvelle gamme » Bull. Ce revirement rapide démontre d'ailleurs que la direction SEMS n'a jamais cru aux briques. Nos partenaires de travail sont donc maintenant les équipes *mainframe* de CII-HB, qui étudient en particulier sur un bus à message, et aussi les équipes de recherche de Bull (Gérard Roucairol, Roland Balter).

Là non plus, on n'aboutit pas à grand-chose faute d'objectifs clairs et partagés, par manque de vision du management Bull cette fois encore.

La licence SM90 : un moment clef

Fin 1982, la direction SEMS signe une licence avec le CNET sur la SM90 et

14Comme quoi, même à l'heure des *big data, des épisodes relativement récents disparaissent de notre mémoire commune.*

fait ainsi, sans vraiment mesurer son importance, le choix UNIX qui va se révéler structurant et ce jusqu'à aujourd'hui. Le site grenoblois, maintenant intégré dans ATOS, a dû à mon sens sa survie à son étiquette UNIX… et aux capacités de survivants de ses employés.

La SM90 est une machine conçue par le CNET, basée sur le processeur Motorola 68000 et un SM-bus propriétaire ; elle fonctionne avec un portage d'UNIX V7.

Jean-Pierre Brulé, ancien P.-D.G. de CII HB de 1972 à 1981, indique dans son livre *L'informatique malade de l'état* que l'État impose à Bull d'acheter auprès du CNET un 9e système d'exploitation, accentuant ainsi la confusion. À mon avis, cette intervention étatique a en fait été, sans que cela ait été fait exprès, salutaire ! On a là un exemple de décision clef prise pour des raisons complètement tactiques et sans aucune vision sous-jacente.

En mars 1983, un directeur technique issu de la recherche, Jean-Claude Chupin, dynamique et sympathique, est nommé pour piloter la « Nouvelle Gamme ». Pendant quelques mois, des études pour ce faire perdurent, mais c'est la licence SM90 qui va en fait être le fil conducteur du futur. SEMS va *de facto* choisir de ne faire ni un microprocessur en circuit intégré, ni une machine 32-bit, mais au contraire d'abandonner les matériels propriétaires et d'accrocher son destin à UNIX et aux microprocesseurs Motorola pour des années.

Officiellement, en octobre 1983, un communiqué tombe : « *Le groupe Bull, après analyse en profondeur des données techniques et des contraintes du marché, a décidé en juin dernier de ne pas entreprendre la mise au point d'une gamme totalement nouvelle et intégrée de mini-ordinateurs* ». Suite à cette décision, François Michel démissionne de son poste de P.-D.G. de SEMS et va se consacrer à la société Copernique qu'il a créée quelques années auparavant. Il est remplacé par Georges Grunberg.

La mission confiée à Échirolles est d'industrialiser la SM90. Contrairement à l'hypothèse de René Chevance[15], nous étions tous très contents de cette décision et nullement accrochés à l'impasse des briques ou de la nouvelle gamme. Enfin, on faisait quelque chose qu'on pourrait vendre ! Néanmoins, il nous a fallu deux ans pour faire un produit, puisque le SPS7, dont Michel Véran était le chef de projet, est sorti seulement en 1985. La première machine 32-bit, équipée d'un 68020, apparaîtra, elle, fin 1986.

UNIX oui, mais comment ?

Le portage UNIX V7 n'était plus pertinent sur le marché, trop ancien pour un objectif fixé à 1985. Rapidement, notre idée du Graal à viser est un UNIX System V avec une pile TCP/IP de BSD : System V parce que c'est plus industriel

15 Voir le texte de René Chevance, déjà cité (note 4).

et facile à vendre à la direction, et la pile BSD parce qu'elle marche.

À Échirolles, à part l'équipe qui développe des protocoles ISO/OSI bien sûr, nous sommes résolument pro-TCP/IP, en partie par clarté de vision des équilibres stratégiques, et en partie parce que ça nous arrange d'enfourcher ce cheval. Il va donc falloir refaire le portage[16] de System V, ce qui va nous servir de formation. Dans ces années-là en effet, un portage UNIX, dans l'esprit des managers, ne peut être fait que par un gourou, de préférence américain. Nous aurons beaucoup de mal à convaincre notre direction que des Français de province peuvent aussi le faire pour moins cher. Il faudrait lister ici les développeurs clefs de ce premier portage, mais j'ai trop peur d'en oublier.

Comment vendre UNIX en interne ?

On aura compris qu'UNIX a peu de défenseurs dans Bull. La direction a eu la main forcée par l'État actionnaire. Le corps commercial craint pour son parc de produits propriétaires DPS6, 7 et 8 et autres Questar 400, et aller chasser en hors-parc est plus difficile : dans un premier temps, il s'agit seulement de distribuer en OEM (Original Equipment Manufacturer) cette machine sur le marché scientifique. Il n'est pas question de cannibaliser la base Bull sur le marché de la gestion.

Les architectes « parisiens » pensent[17] soit qu'UNIX est un non-sujet, soit qu'UNIX a bien des faiblesses et doit être « amélioré » dans les dimensions transactionnelles et de tolérance aux fautes[18].

Les concepteurs des *mainframe* aux États-Unis et en Europe ont, il faut bien le reconnaître, presque tout inventé en informatique dans les années 1960-1970. La mini-informatique n'est qu'une version simplifiée de la grande informatique. UNIX n'apporte quasiment aucun nouveau concept vraiment important. Il est donc normal qu'il soit regardé de haut par certains, ou qu'on rêve au « mainframe UNIX ». Il a en effet nombre de faiblesses :

- Le système de fichiers n'est pas robuste (« journalisé »). À chaque crash, il faut le réparer longuement à coup de *fsck*.
- Il ne marche pas sur un multiprocesseur symétrique.
- Il ne connaît pas la notion de transaction.
- Il n'est pas en temps réel.

16 N.D.E. : le portage est le fait de « porter » un système dans un environnement matériel qui ne lui est pas d'origine. Il s'agit d'une adaptation matérielle et/ou logicielle.

17 C'est une simplification, ils ne pensent bien sûr pas tous la même chose.

18 Sur Trix en particulier, voir le texte de René Chevance, déjà cité (note 4).

- La robustesse et la reprise sur erreurs sont négligées.

Il n'y a aucun antagonisme personnel avec tous ces architectes « parisiens », car nous sommes tous unis par le virus (pas si courant) de l'architecture, et beaucoup sont devenus des amis. Mais chacun défend loyalement sa boutique. René Chevance est de culture CII et Transac ; Jean-Jacques Guillemaud, Jean-Jacques Pairault et Jean Papadopoulo sont de culture DPS7.

Seul l'établissement d'Échirolles a vraiment intérêt à ce qu'UNIX dans Bull marche techniquement et commercialement, car il voit là une bannière à sa mesure. La logique d'établissement dans un grand groupe distribué géographiquement est très puissante, que ce soit à Bull ou ailleurs. Tout site périphérique, dans un groupe Bull qui, à partir de 1986, est en décroissance permanente, est menacé de fermeture. Pour survivre, il lui faut une mission et un marché. Bull Échirolles marche bien dans ce contexte-là ; en général, on nomme alors un patron local qui a à cœur de développer son pré carré. Dans les phases dépressives, l'établissement est coupé en morceaux, rattachés à diverses hiérarchies parisiennes. Aucune dynamique locale de groupe ne peut émerger ; on se borne à survivre. Notre mission doit de plus être à notre mesure, avec des produits assez pertinents pour être vendus ; mais tout projet technique doit être assez petit pour pouvoir être développé localement. Si on élabore un projet ambitieux, il sera inéluctablement aspiré par « Paris ». Ces considérations de bon sens biaisent évidemment nos recommandations techniques, au grand dam de certains. En un mot, la bannière UNIX est à notre mesure, il nous faut la développer et la garder. Un autre facteur humain joue en notre faveur. Les établissements parisiens ont une culture délétère pour UNIX ; seuls des « cousins de province » peuvent s'en saisir. Ce genre de raisonnement n'a rien de spécifique à Bull ; pour faire de l'UNIX sérieusement, IBM a dû créer un site à Austin au fin fond du Texas, loin des anticorps de MVS et de l'AS/400. Les établissements HP et Sun à Grenoble ont connu les mêmes soucis d'éloignement provincial.

Les faiblesses techniques mentionnées ci-dessus nous semblent surmontables ; le système de fichier robuste et le multiprocesseur viendront en leur temps (après 1992 !) et les transactions ACID à la mode *commit* à 2 phases ne rentreront jamais dans l'OS. Les SGBDR et autres Tuxedo géreront cet aspect et le Web a dévalorisé les transactions ACID du fait même de ses faiblesses en tant que réseau[19]. La loi de Moore rendra acceptable l'aspect non-temps réel.

Notre crédibilité technique et commerciale auprès de la direction étant faible, une partie de billard à trois bandes est plus efficace. C'est ce qui nous a amenés à nous appuyer sur une coopé-

19 Il s'avérera d'ailleurs impossible dans les années 2000 de standardiser un WS-Transaction.

ration européenne avec d'autres francs-tireurs comme nous, pour qu'on parle positivement d'UNIX dans les cocktails de l'informatique (voir plus bas).

UNIX distribué : une impasse féconde

Ce paragraphe est quelque peu une digression menée à la lumière de nos efforts de l'époque sur le thème des systèmes distribués. La morale me semble intéressante.

Une recherche de valeur ajoutée par rapport à l'UNIX de « tout le monde » est nécessaire dans un monde ouvert. On se heurte bien sûr au fait que toute différentiation propriétaire est mal acceptée par le client, qui veut précisément éviter cet enfermement. On recherche donc une différentiation « à plusieurs ». Un des thèmes les plus porteurs dans ce milieu des années 1980 est l'UNIX distribué. Pour les architectes d'OS, il s'agit d'étendre les abstractions de l'OS sur un réseau de machines. Pour les architectes réseau, comme notre collègue Michel Habert, il s'agit bien sûr d'étendre les abstractions du réseau dans les machines. « *The network is the computer* », le slogan de Un[20] peut d'ailleurs être entendu des deux manières. On se heurte vite au fait que l'abstraction du processus est peu distribuable et que la seule abstraction accessible est le fichier.

Pour notre part, nous explorons la Newcastle Connection, pendant que Sun est en train d'inventer NFS (Network File System), qui aura un succès certain et sera rapidement indispensable à tous les UNIX du marché. Néanmoins, du fait de son aspect « sans état », NFS ne respecte pas complètement la sémantique du fichier UNIX, en particulier les verrous.

Dans le cadre de son système distribué DCE, OSF introduira DFS, basé sur une technologie Transarc et respectant, au prix d'une certaine complexité, la sémantique UNIX. DCE comportait aussi d'autres mécanismes de distribution, un RPC et un répertoire de nommage. Tout ça n'a eu aucun impact commercial et a été balayé par le Web, comme la multitude de systèmes distribués concurrents.

Le Web, à côté de ses énormes qualités de standardisation, a des propriétés particulières, plutôt médiocres, en termes de performance et de fiabilité. Il est impossible de reproduire sur ce réseau la sémantique d'une machine ou même d'un réseau local. Le Web a *de facto* imposé une sémantique relâchée adaptée à ses défauts (pas de transactions ACID, pas de verrouillage). Ce sont donc les applications qui se sont adaptées au Web, et pas l'inverse.

Néanmoins, des clusters de machines UNIX ou Linux forment la base des centres de calcul des grands acteurs de l'Internet qui constituent le *cloud*, et aussi de nombre de supercalculateurs comme Bull Sequana. Ces machines, du fait de

20 Paternité attribuée en France à Hubert Zimmermann.

leurs spécificités, de leur petit nombre et de leur fonctionnement en mode service, peuvent se permettre d'avoir des matériels et des logiciels « propriétaires ». Ces *mainframe* du XXIe siècle sont donc des variantes d'UNIX distribuées sur une interconnexion locale. Ce sont eux qui servent à réaliser la mutation de l'informatique vers « tout *as a service* ».

Morale de l'histoire : TCP/IP et HTTP ont remodelé l'architecture informatique mondiale « par le bas », du fait de leur simplicité robuste, mais aussi de leurs limitations. La sémantique abstraite ne gagne pas toujours.

La saga de la standardisation UNIX

Fin 1984 émerge la problématique du standard UNIX. Les cinq constructeurs européens créent un groupe informel dénommé BISON, pour Bull – ICL – Siemens – Olivetti – Nixdorf. Jacques Febvre et moi sommes les représentants de Bull. C'est une opportunité incroyable pour nous de pouvoir influer à ce niveau.

Définir un standard UNIX

Il s'agit de définir un standard de programmation UNIX. Cet effort va donner naissance au consortium X/Open. En effet, à cette époque, il commence à y avoir des divergences entre les différentes variantes d'UNIX : celle d'AT&T, celle de Berkeley et celles de chaque constructeur, puisque tout le monde veut faire de l'UNIX (sauf IBM, en tout cas avant 1990).

Sun est dès 1985 le leader du marché UNIX avec ses stations de travail 68020 puis SPARC. À ce titre, il n'a aucun intérêt à favoriser une standardisation des APIs. Il est plus pertinent de promouvoir leur UNIX, SunOS[21], comme standard. Dans ces années-là, IBM a décidé de faire l'UNIX sans en faire, avec une machine ridiculement bridée. Ils sont donc inactifs sur ce sujet. HP et DEC sont tiraillés entre leur base propriétaire (MPE et VMS) et leurs équipes UNIX, et soufflent le chaud et le froid selon les moments.

Les cinq Européens, assez en avance sur leur temps et sous le regard goguenard des Américains du secteur, vont donc élaborer des documents standardisant l'interface de programmation d'UNIX. Ce sera un beau succès et tous les constructeurs finiront par se rallier à ces spécifications en 1990, sous la pression des utilisateurs qui sont les victimes de la guerre des UNIX menée par les grands constructeurs.

La guerre des UNIX à partir de 1988 oppose Sun, le leader du marché, associé à AT&T et les autres (HP, DEC puis IBM ayant finalement rallié les Européens). Le noyau étant stabilisé, la guerre a lieu autour des extensions et améliorations :

21 SunOS sera renommé Solaris en 1991.

le système de fichiers journalisé, le *multi-thread*, l'aspect multiprocesseur etc.

On a vite découvert qu'un même API implémenté par plusieurs constructeurs ne permet pas une vraie portabilité du code source des applications du fait d'effets de bord difficilement maîtrisables. Pour avoir un vrai catalogue d'applications, il faudrait un standard binaire ABI en plus du standard programmatique (API). Des efforts pour élaborer un ABI Motorola et Intel ont eu lieu, mais les difficultés techniques ont rapidement démontré que ce problème est insoluble en pratique[22]. Il faut une implémentation de l'OS, qui crée des ABIs *de facto.* Cette réalité s'impose encore aujourd'hui : Google Android, Microsoft Windows, iOS et MacOS chez Apple sont des implémentations d'un OS, pas des spécifications, et néanmoins la portabilité d'une version à l'autre comporte encore des aléas.

La solution serait donc de construire une « base de source » avec une couche de portabilité sur les différents processeurs cible. C'est le projet OSF/1 auquel Sun s'oppose de toutes ses forces. L'Open Software Foundation (Oppose Sun Forever selon le P.-D.G. de Sun) est créée en 1988 par Bull, DEC, Hitachi, HP, IBM, Nixdorf et Siemens (pour l'essentiel) ; elle est basée à Cambridge, près de Boston. Le projet OS se nomme OSF/1, mais il y a d'autres projets évoqués par ailleurs. DEC, HP et IBM bataillent pour imposer des morceaux de leur code source UNIX. Bull, qui n'a rien d'original en termes d'UNIX, fournit une équipe de développement à Cambridge : Gérard Meyer la supervise et observe les évolutions. Le noyau OSF/1 repose sur le micronoyau Mach issu de CMU (comme MacOS et iOS chez Apple) et incorporera entre autres la pile TCP/IP de BSD et le JFS d'IBM. Au final, seul DEC commercialisera OSF/1 en 1992 sur ses machines Alpha sous le nom Tru64. HP et IBM préféreront cultiver leur jardin HP-UX et AIX, une fois la menace Solaris écartée.

OSF a un centre de recherche, et celui-ci a une antenne grenobloise, à l'initiative de Jacques Stern qui est en 1988 le président de Bull. Cette antenne est située à proximité du centre de recherches Bull – IMAG sur le campus grenoblois (Roland Balter), et est dirigée par Jacques Febvre, détaché de Bull à cette occasion. Ce centre a réalisé pour le compte d'Apple la première version de ce qui est aujourd'hui Mac OS, basé sur un micronoyau Mach qui sous-tend une personnalité UNIX et une personnalité Apple.

La guerre des UNIX s'apaisera vers 1994, sous la pression de l'ennemi commun Microsoft. X/Open et OSF fusionneront en 1996 dans The Open Group, lequel ne va d'ailleurs rien faire de significatif.

Pour nous, il a été passionnant de comprendre comment faire converger les

22 Le projet ANDF de OSF est une tentative de définition d'un langage intermédiaire normalisé, entre les langages de haut niveau (comme C ou Fortran) et le langage assembleur spécifique à une architecture matériel ; son échec a démontré l'inanité de tels efforts.

besoins des partenaires X/Open, chacun avec ses intérêts particuliers. Les discussions dans les comités de normalisation sont une expérience unique qui permet d'imaginer les arcanes de la diplomatie. Cela a pris des années, mais on a abouti à la publication en 1985 puis 1987 des divers tomes du *X/Open Portability Guide*. Le *design by committee* fonctionne donc parfois.

Une idylle franco-italienne

En 1987, Bull rachète Honeywell Information Systems (HIS)[23], qui commercialise en partie les mêmes machines DPS. Pour ce qui nous concerne, le point essentiel est que la filiale italienne de HIS, installée à Pregnana Milanese, a développé et vend une gamme UNIX fort semblable à la nôtre. La direction demande très logiquement de faire converger les deux gammes dans le cadre d'un programme appelé (comme toujours) New Common Line. Pour ce faire, elle crée une *joint venture* franco-italienne avec tout pouvoir et un leader italien charismatique, Lucio Pinto.

L'aspect humain est ici plus intéressant que l'aspect technique. Les deux établissements d'Échirolles et de Pregnana sont extrêmement semblables, avec la même culture de « survivants de province » dans un groupe international. La seule différence est que, pour répondre ce bon mot, les « Français sont des Italiens de mauvaise humeur ». Qui se ressemble ne s'assemble pas forcément, les doublons sont innombrables en termes de matériel, micro-logiciel et logiciel. Pour une fois, la finesse de la direction et la nécessité pour les parties de trouver un compromis pour survivre va permettre de se partager le travail pendant plus de dix ans.

En 1987, il s'agit de renouveler la gamme UNIX avec un monoprocesseur d'entrée de gamme et une variante[24] de multiprocesseur en milieu de gamme. Le partage du travail désigne Échirolles pour faire le petit système et Pregnana le gros. Mais l'OS bien sûr, qu'on appellera BOS (Bull Open System), est unique. À cette époque, le marketing a la passion de « bulliser » les noms, alors que la marque UNIX, bien plus connue, serait utilisable et plus facile à vendre à notre avis.

On devrait donc logiquement faire un seul portage de l'UNIX avec deux cibles qui diffèrent légèrement au niveau des entrées/sorties. Mais il y a deux équipes UNIX des deux côtés des Alpes, qui ont à peine trois ou quatre ans d'âge et ne sont pas prêtes à être euthanasiées sur l'autel transalpin. Les deux chefs de projet (étant moi-même l'un d'eux) vont donc s'échiner à prouver que, faute de pouvoir récupérer tout, il vaut mieux que chaque établissement fasse son portage.

23 Financièrement, cette fusion s'étale de 1986 à 1988.

24 Ce n'est pas un vrai SMP (Symetric MultiProcessor) à mémoire commune, mais une variante originale avec mémoire locale et mémoire commune, sortie du cerveau fertile du génial architecte italien Angelo Ramolini.

Les arguments techniques sont faibles, la direction elle-même n'est pas dupe, mais c'est le meilleur moyen de ne pas faire désespérer Billancourt ; et finalement, bien que techniquement irrationnel, ce choix s'est révélé le bon. Les machines sont sorties avec moins de six mois de retard, ce qui est un exploit dans l'univers informatique de l'époque.

À partir de 1985, le cœur d'UNIX n'évolue plus guère. Seul manque l'aspect multiprocesseur symétrique qui viendra dans la décennie suivante.

Épilogue : UNIX multiprocesseur

Bull a déjà fait deux tentatives pour introduire une machine RISC dans sa gamme UNIX, grâce à des collaborations avec de petites entreprises de la Silicon Valley, d'abord avec Ridge commercialisé sans grand succès sous le nom SPS9 fin 1984, puis lors d'une alliance avec MIPS qui a échoué en 1991. Réalisant alors que le problème du catalogue d'applications est insoluble sans être adossé à un acteur majeur, Bull négocie avec HP et IBM et décide finalement en 1992 d'une alliance avec Big Blue, son ennemi héréditaire.

Il s'agit de construire ensemble une gamme de machines UNIX multiprocesseur à base de processeurs PowerPC. IBM s'est résigné à cette époque à lever le veto, autorisant dès lors ses équipes à concevoir de grosses machines UNIX qui vont concurrencer ses gammes propriétaires. Techniquement, le problème est de disposer d'un UNIX multiprocesseur efficace. C'est un travail considérable que de poser dans le code UNIX l'ensemble de verrous permettant de fonctionner avec beaucoup de processeurs (huit pour commencer).

Débute alors une longue revue technique entre nous et les architectes IBM d'Austin, une brochette de *fellows* IBM brillants, sympathiques et pas du tout arrogants (à notre grande surprise). Nous, côté Bull, privilégions un portage sur PowerPC de OSF/1 qui est raisonnablement efficace en multiprocesseur et contient nombre de technologies IBM (JFS etc.). De plus, un certain nombre de développeurs Bull ont participé au projet OSF/1 et ont donc une compétence sur ce sujet. IBM, malgré son implication significative dans les développements OSF/1, hésite et envisage de partir du code AIX et de le rendre multiprocesseur. Sans surprise, la décision tombe finalement du côté AIX. Sa modification prendra quelques années[25] et s'étalera même sur une décennie, car quand on augmente le nombre de processeurs (192 cœurs de nos jours[26]), il faut raffiner la finesse des verrous.

25 Talbot, J., 1995. « Turning the AIX Operating System into an MP-capable OS », sur le site de USENIX [URL : www.usenix.org/publications/library/proceedings/neworl/full_papers/talbot.ps].

26 Sans compter le *multithread* simultané à 8 voies interne à chaque cœur !

Moralité : c'est le plus gros qui gagne, et la rationalité technique est parfois voilée par l'attachement des équipes à leur bibliothèque de code.

Conclusion

À travers l'histoire d'UNIX à Bull Grenoble durant la décennie 1980, on traverse en fait l'histoire de l'informatique mondiale à cette époque, avec nombre de leçons réutilisables de nos jours.

On notera avec intérêt que pour l'essentiel, l'impact sur l'informatique actuelle de nos activités de l'époque est dû à des effets de levier à travers des partenaires plus importants que Bull : IBM, DEC et HP pour X/Open, Apple pour MacOS via OSF, IBM pour AIX.

Encadré

Unix et les systèmes ouverts dans Bull, avant l'Internet

Michel Élie
Responsable du département Architecture réseau et standards de systèmes distribués de Bull (1981-1989)[1].

Philippe Picard
Président de l'Association pour l'Histoire des Télécommunications et de l'Informatique (AHTI).

Cette contribution se propose de résumer les grandes étapes de la stratégie du constructeur Bull dans le domaine des systèmes ouverts et plus particulièrement à propos du système Unix (en complément du témoignage de Jacques Talbot, « Unix vu de province », publié dans ce numéro). Elle insiste sur l'importance pour Bull d'avoir disposé d'une offre cohérente de matériel et de logiciel, malgré la diversité de ses composantes, dont Unix. Elle revient sur la chronologie de l'évolution de cette offre et de son environnement : usagers, autres constructeurs, Direction Générale des Télécommunications, Commission Européenne…

Elle repose largement sur des documents disponibles sur les sites historiques de la Fédération des Équipes Bull (FEB) [1] (en particulier Chevance, 2004 ; Guédé, *n. d.*), de l'Association pour l'Histoire des Télécom et de l'Informatique (AHTI) [2], ainsi que la documentation personnelle des auteurs.

Le point de départ : 1984

En 1984, l'offre de CII Honeywell Bull traduisait l'histoire mouvementée de Bull, après l'achat de Bull par General Electric en 1964, celui de Bull General Electric par Honeywell en 1970, la fusion de CII et Honeywell Bull en 1976, et l'intégration dans Bull de CIT Transac, filiale bureautique d'Alcatel en 1982.

1 Il est possible d'adresser des retours sur ce texte à michel.elie@wanadoo.fr.

L'offre de Bull était articulée autour de cinq lignes et systèmes d'exploitation (OS) stratégiques.

Trois lignes partageaient la dénomination commerciale GCOS : GCOS6, GCOS7, GCOS8. À l'origine, le nom GECOS (devenu GCOS) avait été créé pour désigner le système d'exploitation du système GE-635 introduit en 1964. L'acronyme signifiait General Electric Comprehensive Operating System. Chaque ligne GCOS avait une origine historique différente : Honeywell avec une équipe R&D (recherche et développement) à Boston pour GCOS6, Bull à Paris pour GCOS7, General Electric à Phoenix pour GCOS8. Les équipes de R&D de chaque ligne ont été maintenues sur leur implantation historique. On notera que malgré leur dénomination commerciale commune, les trois lignes GCOS étaient incompatibles et n'assuraient ni interopérabilité ni inter-portabilité. Seules les télécom permettaient un minimum d'échanges entre les trois systèmes.

Le système CTOS était un système multipostes développé par Convergent Technology. CIT Transac en avait acquis la technologie et les droits de distributions, repris par Bull avec les restructurations industrielles.

Le dernier OS était le système DNS, le logiciel de télécom conforme à l'architecture ISO/DSA et qui gérait les « couches basses » des télécom (protocoles X21 et X25, Ethernet, le transport, etc..) et les multiples types de terminaux. Le logiciel était implanté sur un frontal Datanet (commun à GCOS7 et GCOS8) et directement sous GCOS6 et CTOS.

Malgré le lointain héritage de Multics, encore largement utilisé par les équipes de R&D pour les échanges transatlantiques, la culture Unix était quasi absente chez Bull.

Axes stratégiques

Dès son arrivée à la tête de Bull, en 1982, Jacques Stern a compris que Bull n'aurait jamais la taille lui permettant de jouer un rôle mondial, quelle que soit la qualité de son offre de matériel et logiciel propriétaires qui était plus qu'honorable. Le premier coup de semonce fut observé dès la fin des années 1980 avec l'offre CTOS qui n'a pas résisté au monde plus ouvert « *wintel* » (c'est-à-dire un monde dominé par le système Windows de Microsoft et les microprocesseurs d'Intel), malgré ses qualités intrinsèques et sa bonne intégration dans l'offre Bull.

L'intérêt des systèmes ouverts était clairement exprimé par le marché (même s'ils ont longtemps été considérés comme utopiques) avec deux objectifs : la liberté de

choix vis-à-vis des fournisseurs et la richesse des applications disponibles. Le dilemme pour Bull était qu'il fallait à la fois soutenir et faire évoluer l'offre existante des systèmes propriétaires tout en préparant l'avenir avec une offre de systèmes ouverts. D'où l'engagement de Bull sur deux axes stratégiques en faveur des systèmes ouverts : les communications et Unix.

La normalisation des réseaux et la position de la Commission Européenne

Bull menait avec Honeywell une stratégie visant à empêcher IBM d'imposer son architecture de réseaux SNA (Systems Network Architecture) comme un standard *de facto* du marché. Cette stratégie s'appuyait sur les concepts techniques mis en œuvre dans sa propre architecture de systèmes distribués : DSA (Distributed System Architecture). Sa position fut rejointe dès 1978 par la plupart des constructeurs européens regroupés dans l'ECMA (European Computer Manufacturer Association, basée à Genève).

Au milieu des années 1980, le protocole du modèle Internet, TCP/IP, n'était pas encore important : sa généralisation sur le réseau Internet aux États-Unis ne date que de 1983. La distribution gratuite des versions Unix BSD 4.2 et suivantes livrées à partir de 1983 par l'Université de Berkeley, incluant le support de TCP/IP, accéléra le développement de l'Internet, d'abord aux États-Unis, puis, peu à peu, en Europe, chez les universitaires, comme une alternative à des développements coûteux basés sur les normes internationales OSI d'interconnexion de systèmes ouverts.

Il faut bien avoir à l'esprit le contexte de politique industrielle européenne dans lequel on se situait, bien décrit début 1987 dans un numéro spécial de la *Revue d'économie industrielle*, « Les nouvelles industries de l'information et de la communication », sous la direction de Marc Humbert & Laurent Gille, en particulier dans l'article de Emmanuel de Robien (1987) alors directeur de la prospective de Bull. Un mouvement en faveur du développement de l'industrie européenne informatique avait été lancé par Étienne Davignon, vice-président de la Communauté Économique Européenne (CEE) en créant la « Table ronde des 12 » (GEC, ICL, Plessey, AEG, Nixdorf, Siemens, Bull, Olivetti, CGE, Thomson, Stet, Philips).

La CEE, puis l'Union Européenne (UE) étaient favorables à la normalisation de l'interconnexion des systèmes ouverts (OSI). Elles ont favorisé une coopération régionale pour en développer des implémentations prototypes portables sous Unix dans le cadre du programme ESPRIT (« European Study Program in Information Technology », programme stratégique européen pour la recherche en technologie de l'information).

Engagement de Bull vis-à-vis d'Unix

Les équipes de la direction Réseau de Bull ont joué un rôle important dans ces projets, en coopération avec leurs partenaires GEC, ICL, Olivetti, Siemens, Philips. Bull fut également très actif dans les instances internationales pour promouvoir le développement des systèmes ouverts, et faciliter la propagation d'Unix :

- X OPEN créée en 1984 par Bull, ICL, Siemens, Olivetti, Nixdorf produit le X/Open Portability Guide et fusionne avec l'OSF en février 1996, pour donner naissance à l'Open Group.

- En 1988, Bull fit partie des fondateurs de l'OSF (Open System Fondation) avec Apollo Computer, Digital Equipment Corporation, Hewlett-Packard, IBM, Nixdorf Computer, et Siemens AG. L'objectif était de diffuser entre autres le code source d'une version de référence d'Unix.

Une opportunité : le SM 90

Dès 1980, le Centre national d'études des télécommunications (CNET), en partenariat avec l'Institut national de recherche en informatique et en automatique (INRIA), a lancé la conception d'une machine destinée à des applications techniques pour le réseau téléphonique. Ce fut le SM 90, une machine Unix architecturée autour d'un bus spécifique, le SM-Bus permettant l'adjonction facile de coupleurs spécifiques. Un Groupement d'intérêt public, le GIPSI-SM 90, fut créé entre l'INRIA, le CNET et Bull afin de promouvoir les applications sous Unix. Le GIPSI fut présidé par Jean-François Abramatic, qui deviendra en 1995 président du World Wide Web Consortium. De nombreux industriels prirent le brevet du SM 90 : Bull, Thomson Telephone, TRT, CSEE, ESD Dassault, SMT-Goupil et Telmat. Le succès commercial de SM90 fut surtout dans le domaine de l'informatique technique dans le réseau de France Télécom plutôt que dans la bureautique. Dans ce domaine, le CNET avait lancé un projet de bureautique interne nommé Smartix, sur la base de SM 90 et d'un serveur Multics (Picard, 2017, annexe II).

Unix avait déjà acquis une grande notoriété aux États-Unis du fait de la décision d'AT&T (forcée par un décret anti-trust) de le diffuser largement et gratuitement dans les milieux universitaires. Le marché Unix en France était encore balbutiant : milieux de chercheurs informatiques et autres « convertis », stations de travail… Quelques grands clients ont amorcé le marché, par exemple EDF et surtout la DGT pour des applications techniques spécifiques.

Malgré l'offre pléthorique en lignes de produit, Bull décida de démarrer une offre Unix en industrialisant le SM 90 sous le nom de SPS 7 et en achetant à AT&T la licence d'Unix, assez onéreuse à cette époque. Des améliorations devaient lui être apportées pour satisfaire les besoins d'un marché européen, dont une difficile internationalisation pour supporter des jeux de caractères nationaux, ce que ni ses concepteurs ni les chercheurs de Berkeley n'avaient pris en compte, bien que ceux-ci aient été proches du centre de recherche de Xerox qui maîtrisait parfaitement ces questions.

Bull a développé des coupleurs informatiques *ad hoc* (X21, X25, Ethernet) et les couches ISO/DSA pour intégrer l'offre Unix dans l'architecture de Bull. Une coopération avec un téléphoniste (Jeumont Schneider) a même conduit Bull à développer des coupleurs en technologie de téléphonie RNIS, puis des protocoles spécifiques aux télécoms (comme le code CCITT n° 7).

Le décollage du marché Unix

Le SPS 7 a rapidement trouvé ses limites et Bull a décidé de développer une gamme Unix en profitant de développements en cours chez Bull Italie à base de Motorola 68000 et de Multibus. Il a été remplacé par les machines de la ligne DPX 2 disponible au début des années 1990 qui connurent un succès très honorable. En dehors des milieux universitaires et des marchés spécifiques comme les télécom, Unix correspondait à un véritable souhait des utilisateurs de sortir du carcan des systèmes propriétaires des grands constructeurs. Des outils nouveaux sont en outre apparus, rendus disponibles sur Unix, à commencer par les Systèmes de gestion de bases de données (SGBD) relationnels (Ingres, Informix et surtout Oracle).

Les machines de la ligne DPX 2 ont été utilisées pour des grands déploiements techniques, par exemple chez France Télécom pour le système de collecte en temps réel des tickets de taxation du réseau (les célèbres « fadettes »), les messageries vocales du réseau mobile, le système de test des lignes, etc.

Les partenariats et alliances diverses

On sait qu'un des segments importants au démarrage commercial d'Unix fut celui des stations de travail (en particulier de Sun Microsystems). Bull tenta une incursion sur ce marché par des accords de distribution avec Ridge, vendu sous le nom SPS 9, sans grand succès. L'offre était moyenne face aux grands concurrents, mais surtout elle s'adressait à un segment de marché que Bull ne connaissait pas bien.

Par ailleurs, Bull a recherché des alliances technologiques pour l'aider à monter en gamme :

- avec MIPS, pour étudier les architectures multiprocesseurs ;
- avec Tolerant Systems, pour développer un système basé sur Unix et sécurisé (projet Trix).

Finalement les divers projets envisagés avec ces partenaires ont été abandonnés avec la décision stratégique d'alliance avec IBM.

L'alliance avec IBM

En effet, à partir de 1991, la Direction Générale de Bull a pris conscience de l'impossibilité de « suivre le mouvement » du marché Unix avec ses propres moyens et compte tenu de l'impasse du 68000. Elle a donc recherché une alliance avec un « grand ».

Le débat s'est rapidement concentré sur le choix entre Hewlett Packard et IBM. C'est finalement IBM qui a été choisi, l'un des critères étant la complémentarité technologique et la valeur ajoutée pouvant être apportée par Bull qui a été chargé en particulier du développement des télécom (coupleurs et couches ISO) et de la coopération dans le domaine de multiprocesseurs. Ce choix a curieusement pris un tour politique, Abel Farnoux, conseiller spécial du premier Ministre de l'époque, Edith Cresson, étant très partisan de HP. Les patrons de Bull ont heureusement pu imposer leur choix. Le sujet avait été largement débattu lors d'une participation d'Edith Cresson à une réunion de l'AHTI en juin 2009 où elle était intervenue sur les problèmes industriels et économiques de l'innovation (Cresson, 2009).

Cette alliance, conclue en janvier 1992, a été assez fructueuse, Bull a non seulement assimilé rapidement le système AIX d'IBM inspiré d'Unix, mais a développé des configurations haut de gamme, multiprocesseurs grâce à ses compétences d'études de Grenoble, ainsi que les télécom pour le marché européen. Cependant, à la fin de la décennie 1990, les machines haut de gamme AIX furent nettement distancées par les machines HP et surtout les machines Solaris de SUN pour les applications demandant une forte puissance.

Unix et les systèmes propriétaires

Afin de ralentir l'érosion du parc des systèmes propriétaires ou de piloter la récupération du parc, Bull a mis en place plusieurs familles d'outils. Pour GCOS 7 et

GCOS 8, a été développé un coprocesseur Unix, sorte de frontal fortement couplé avec le CPU propriétaire permettant par exemple d'adjoindre à une application transactionnelle une base de données relationnelle ou un frontal web. Mais aussi pour GCOS 6, a été développé un émulateur sous AIX permettant la récupération du code ancien de très nombreuses applications « métier » ayant survécu au nettoyage de l'an 2000.

Les équipes Unix à Bull

Comme le montre la contribution de Jacques Talbot, l'équipe chargée des développements pour l'industrialisation d'Unix était localisée dans l'établissement de Bull à Échirolles dans la banlieue de Grenoble, et était très concernée par sa survie dans ce site. Elle était la lointaine héritière des compétences de l'équipe Télémécanique qui avait développé la gamme SOLAR. Cette équipe a donc participé, avec une grande compétence et d'autant plus de détermination, aux différents choix techniques et stratégiques de la ligne de produit Unix. Elle a pu pour cela s'appuyer sur une large communauté technique rassemblant des équipes de tout horizon bien au-delà de leur propre entreprise. Ainsi, sans doute pour une des premières fois, est apparu un communautarisme technique susceptible d'exercer une pression sur la stratégie des entreprises. Des liens se sont établis entre équipes techniques d'entreprises concurrentes, aboutissant parfois à la mise en circulation « libre » de certains logiciels développés dans ce cadre.

Chronologie

1982 Accord de licence ATT-Bull.

1984 Fondation de X/Open Group par Bull, ICL, Siemens, Olivetti et Nixdorf, Siemens Olivetti et peu après rejoints par Philips et Ericsson dans le but de développer une spécification commune aux systèmes d'exploitation dérivés d'Unix. Ce groupe publiera à partir de 1985 le *X/Open Portability Guide.*

Introduction par Bull-Sems de la ligne basée sur la SM 90 sous le nom de SPS 7 basé sur la technologie issue du CNET.

Introduction du système Unix RISC de la compagnie Ridge sous le nom de SPS 9, station de travail puissante.

1987 Bull-Transac, Bull-Sems et Bull-Micral sont réunis dans une seule entité Bull-MTS. Création d'une ligne de produits Unix.

1988 Création d'une filiale X3S commune à Bull SA et à Honeywell Bull Italia pour développer ensemble le marché Unix.

1989 Transfert de la maintenance de Multics : de Honeywell-Bull à l'ACTC de Calgary.

1990 Annonce de la ligne DPX 2, offre Unix basée sur 68000 et Multibus2, résultant des développements franco-italiens. Elle incluait une version « internationalisée » d'Unix.

1991 Annonce du Bull Computing Model (de nom de code PurpleWay) un concept d'informatique distribuée basé sur les standards Unix, mais applicable aussi aux systèmes GCOS et aux PC.

Bull arrête ses efforts pour baser ses systèmes Unix sur l'architecture MIPS et se met à la recherche de nouveaux partenaires.

1992 Annonce d'un accord entre Bull et IBM sur AIX.

Bull adopte l'architecture PowerPC (commune à IBM et à Motorola). Bull doit développer un multiprocesseur destiné à être vendu par IBM et Bull. IBM prend 4.5 % du capital de Bull (initialement prévu 5.7 %).

Premiers sites web ouverts en France : INRIA, Cnam, IN2P3…

1994 Annonce du multiprocesseur Escala (né sous le nom de code Pegasus) issu d'une coopération entre Bull Italie, Bull Echirolles et IBM Austin, opérant sous une nouvelle version de AIX.

Sources

[1] Site web de la Fédération des Équipes Bull (FEB) [URL : http://www.feb-patrimoine.com].

[2] Site web de l'Association pour l'Histoire des Télécoms et de l'Informatique (AHTI) [URL : http://www.ahti.fr].

Bibliographie

Chevance R. J. (2004). « Contribution à l'histoire d'Unix chez Bull ». En ligne sur le site Web de la Fédération des Équipes Bull [URL : http://www.feb-patrimoine.com/projet/unix/histoire_unix_chez_bull_12_2004.pdf].

Guédé J.L. (d'après). (*n. d.*). « Bull SPS 7 ». En ligne sur le site Web de la Fédération des Équipes Bull [URL : http://www.feb-patrimoine.com/projet/unix/sps7.htm].

Cresson E. (2009). « Conférence de M^me^ Édith Cresson, ancien Premier Ministre ». *Cahier de l'AHTI,* n° 12, pp. 7-17 [URL : http://www.ahti.fr/cahiers/Cahier12.pdf].

De Robien E. (1987). « Les stratégies de normalisation dans le domaine de l'information ». *Revue d'économie industrielle,* vol. 39, 1^er^ trimestre, pp. 220-227 [URL : http://www.persee.fr/issue/rei_0154-3229_1987_num_39_1].

Picard P. (2017). « La bureautique des années 1980 et le projet Scribe ». *Cahiers de l'AHTI,* n° 22, pp. 17-18 [URL : http://www.ahti.fr/cahiers/Cahier 22.pdf].

Imprimé dans les ateliers d'impression du CNAM
sur un papier agréé FSC/PEFC respectueux de l'environnement.

Cahiers d'histoire du Cnam

La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II

coordonné par Camille Paloque-Berges et Loïc Petitgirard

Dossier : La recherche sur les systèmes : des pivots dans l'histoire de l'informatique – II/II

vol. 7–8

2017 / Second semestre

(nouvelle série)

ISSN 1240-2745