La conversion à Unix Un exemple de prophétisme informatique ?

Laurent Bloch

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.