QWERKS OF HISTORY

Brian Hayes

Let me take you back to 1969, the year of Woodstock and of Neil Armstrong's giant leap for mankind. Nixon was in the White House that year, Elvis was playing Vegas, and the Beatles were recording *Abbey Road*. Bill Gates and Steve Jobs were high school kids. Such a long time ago it was—half a lifetime. In the world of computer technology, 1969 is halfway back to the dawn of time.

And yet, when you look closely at the latest computer hardware and software, it's not hard to find vestiges of 1969. Your shiny new Pentium processor can trace its heritage back to a chip whose design was begun that very year. The Unix operating system, whose offshoots are blooming luxuriantly these days, also has its roots in that flower-power summer. And the first four nodes of the ARPANET, progenitor of today's Internet, began exchanging packets in the last months of 1969. The window-and-menu interface we know so well was still a few years in the future; but, on the other hand, the computer mouse was already a few years old.

One reaction to the longevity of so much early computer technology is admiration for the pioneers of the field. They must have been clear-thinking and far-seeing innovators to get so much right on the first try. And it's true: Giants strode the earth in those days. Hats off to all of them!

But in celebrating the accomplishments of that golden age, I can't quite escape the obvious nagging question: What has everybody been doing for the past 35 years? Can it be true that technologies conceived in the era of time-sharing, teletypes and nine-track tape are the very best that computer science has to offer in the 21st century?

Qwerks

It's no secret that the success of a technology is not always explained by merit alone; there are also qwerks of history and qwerks of fate—named for the most famous example, the *qwerty* keyboard on which I am typing these words. The *qwerty* layout is not the best possible arrangement of the letters—it may even be the worst—but once everyone has learned it, the cost of change is too great. A slightly different kind of qwerk is sometimes cited to explain the ascendancy of the gasoline-fueled automobile engine. Whether or not this powerplant was the best choice at the outset, the argument goes, so much engineering effort was invested in its development that rivals couldn't keep up. By now, the global infrastructure supporting gasoline engines gives them an almost insuperable advantage. The story of color television offers yet another variation on the theme: The broadcast format adopted in the 1950s was chosen not because it gave the highest picture quality but because it was compatible with existing black-and-white TV sets. Even 50 years later, replacing that suboptimal format with the digital HDTV standard has been slow going.

These stories imply a certain mental model of technological evolution. When a new product category appears, there's an initial free-for-all period, in which diverse ideas compete on a more-or-less equal basis. But multiple alternatives cannot coexist indefinitely. Eventually one technology becomes dominant—for whatever reason—and all others are driven out. A latecomer to the market has little hope of breaking in.

The trouble with this neat formulation is that counterexamples are as easy to find as examples. If it were true that an established technology could never be dislodged from its niche, we would still be listening to vinyl LP records. If competing technologies could never co-exist, we would not have three incompatible standards for cellular telephones in the U.S., with a fourth standard prevalent elsewhere. It appears that some technologies remain fluid over long periods, whereas others freeze solid early in their history. How do we tell them apart? What is it about audio recording that has allowed such a profusion of media (wax cylinders, 78s, 45s, 33s, eight-track cartridges, cassette tapes, CDs, MP3s), while keyboards have remained steadfastly faithful to *qwerty*? Perhaps close examination of specific cases and circumstances would explain these discrepancies, but I don't see much hope for a simple, predictive theory of technological evolution.

The realm of computing offers a great many further illustrations of qwerkiness. Here I shall

Brian Hayes is Senior Writer for American Scientist. *Address: 211 Dacian Avenue, Durham, NC 27701; bhayes@amsci.org*

discuss two examples of strangely long-lived technologies, both of them distinguished members of the class of 1969: the Intel family of microprocessors and the Unix operating system.

Eighty-Sixed

The chip that founded the Intel dynasty was designed for a Japanese desktop calculator. The processor, called the 4004, had about 2,300 transistors and operated on four bits of data at a time. The 4004 was followed by the 8008 and the 8080, which were eight-bit processors, and later by the 16-bit 8086. (A transitional morph designated the 8088 was chosen for the first IBM PC, in 1981.) Later came the 80186 and the 80286; then, expanding the data path to 32 bits, the 80386 and the 80486. Just when we thought we understood the naming scheme, the next chip in the series was called the Pentium. There have been several further iterations; the current model is the Pentium 4, with a transistor count of 42 million.

The processors in the "*x*86" series have much more in common than just the corporate logo stamped on the package. (Indeed, chips made by several other companies belong to the same family.) All of the processors since the 8086 are machine-code compatible; that is, a program written for the 8086 in 1978 should run without change on the latest Pentium. This impressive feat of backward compatibility did not come without cost and effort. Through all the generations of processors, the designers have had to preserve the same basic architecture—features such as a set of eight general-purpose registers, a partitioning of the computer's memory space into fixed-size segments, and an instruction set that by modern standards might best be described as *funky*. The instructions vary in length from eight bits to 108, and they include a baffling variety of modes for addressing data in memory.

Fashions in computer engineering have changed since 1969, and the *x*86 instruction set now looks as retro as a pair of wide-wale bell-bottoms. Stylish microprocessors these days are mostly RISC designs—an acronym for *reduced-instruction-set computer*. The instructions are kept simple and few so that they can be executed very fast. This strategy frees up space on the chip for other resources, such as a large set of registers. Uniformity and orthogonality are also prized virtues of RISC designs: Usually, all the instructions have the same length and the same format, and all the registers are interchangeable.

By now, RISC is everywhere in the world of microprocessors—except the *x*86 line. Even Intel has introduced RISC chips, as well as another adventurous new technology called a very-long-instruction-word processor, but these products are not being marketed for mainstream personal computers. One reading of this situation is that Intel has been trapped by its own success. The designers might well prefer to leave behind the qwerks of the past, but the marketplace will not allow them to abandon 25 years' worth of "legacy code" written for the *x*86 chips.

It's interesting to compare Intel's predicament with the experience of Apple Computer, which converted the Macintosh to a RISC architecture in 1994, yet maintained compatibility with older software. The changeover was rather like jacking up a house and moving it to a new foundation while the family inside sat down for dinner. The key to backward compatibility was an emulator, a program running on the new hardware that impersonates the old.

Why couldn't Intel do something similar to liberate their customers from the *x*86 architecture? In fact they did, although the impersonation is done in hardware rather than software. Inside the latest Pentium processors is a core that looks nothing like the earlier *x*86 chips; it is a RISC-like machine with a large bank of registers and a repertoire of simple instructions called *R*-ops (for RISC operations). But this core is entirely hidden from the programmer, who continues to deal with the chip as if it had the usual set of eight general-purpose registers and other qwerks. A decoder on the chip reads incoming *x*86 instructions and translates them into *R*-ops. Most of the translations are straightforward, but a few of the *x*86 instructions give rise to streams of more than 100 *R*-ops.

One might try to argue that the Pentium 4 would be faster without the overhead of translation. The chip real estate dedicated to the decoder

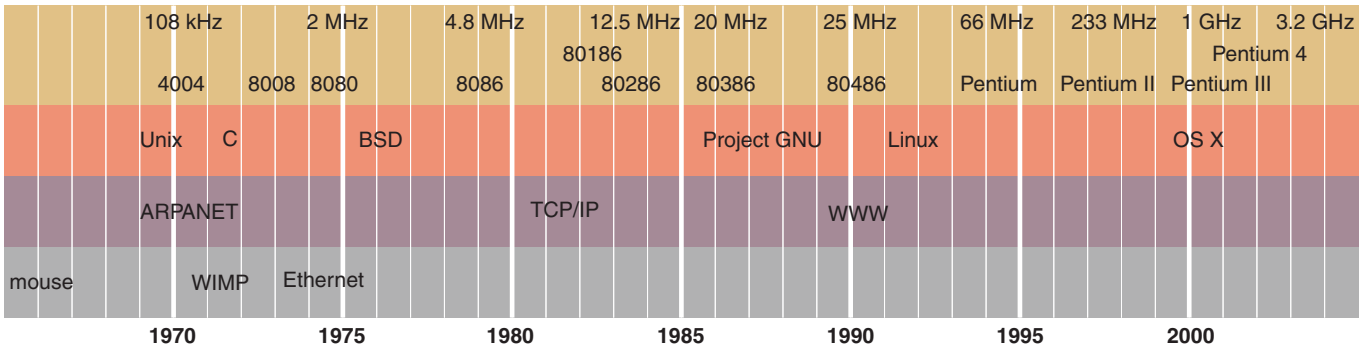

Figure 1. Although the world of computing is ever-changing, a surprising amount of computer technology has roots deep in the past. The Intel "*x*86" line of microprocessors, the Unix operating system and the Internet all trace their heritage to 1969; the WIMP (windows, icons, menus, pointing device) interface is not much younger; and the mouse was invented in 1965. Numbers at the top give clock speeds of the corresponding processors.

could be put to some other use that would improve overall performance. This may be true. On the other hand, users of Pentium 4 systems are not complaining about traveling in the slow lane while the world passes them by; on the contrary, makers of other chips have had to struggle to keep pace. The fastest Pentium has a clock rate greater than 3 gigahertz. It is the gasoline engine of computing.

Unix

Software, as the word itself suggests, is more malleable than hardware, and so one might think it would also be more ephemeral. The history of Unix suggests otherwise.

Unix began as a small-scale, low-budget project of Dennis M. Ritchie and Ken Thompson at Bell Laboratories. It has grown and prospered. Today it is the software of choice for scientific and engineering workstations and for multicomputer grids and clusters; it also runs many Internet servers and other "back office" machines. The Linux variant is installed on several million PCs. Recently the Macintosh has become yet another Unix box. Furthermore, Unix has exerted an unmistakable influence on other operating systems, not least Microsoft Windows. In certain crucial internal structures, such as the organization of files into nested directories, almost all computer systems today have adopted methods that were first introduced in Unix.

The early versions of Unix were written for minicomputers in the PDP series, made by Digital Equipment Corporation. Many of those machines had only a few kilobytes of memory; disk storage was equally scant; the standard means of communicating with the computer was a hardcopy teletypewriter or a text-only video terminal. Looking back from our present abundance of gigabytes and megapixels, it seems remarkable that a complex software system could be made to work at all on such primitive machinery. That it would still be running today—and would be widely regarded as the best of its kind—is astonishing.

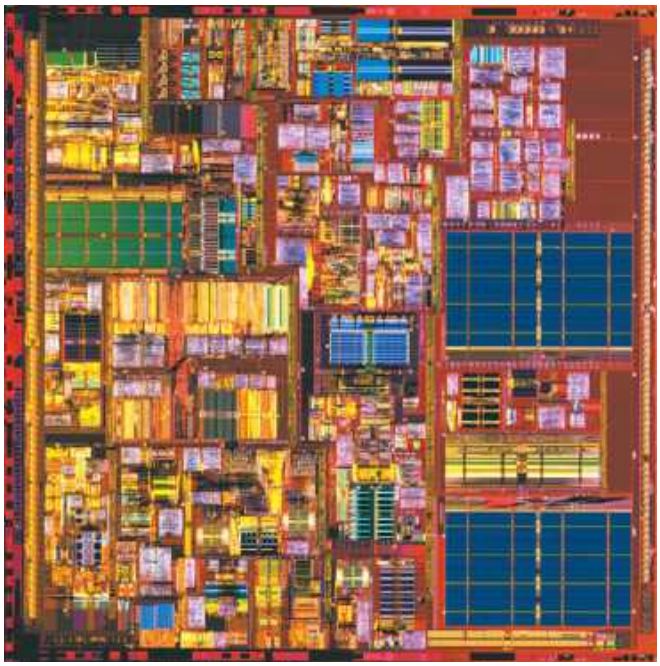

Figure 2. Intel Pentium 4 microprocessor, introduced in 2000, can run programs written in the same instruction set as the Intel 8086, which was announced in 1978. But the Pentium 4 does not execute those instructions directly. They are translated into the simpler commands of a reduced-instruction-set computer (RISC). The chip has some 42 million transistors; it is fabricated with a smallest feature size of 0.13 micron. Photograph reprinted by permission of Intel Corporation.

Of course the Unix of today is not your grandfather's operating system. If you wish, you can still invoke programs from a 24-by-40-character terminal (and some thoughtful people insist that this "command line" remains the best interface), but there are also glitzy window-and-mouse alternatives. Under the hood, the internals of the system have been rewritten over and over—so much so that Unix has to be seen as a kind of standing wave, a pattern that persists even though all of its substance is constantly in flux. Quite possibly, not one line of code has been preserved from the earliest versions. (In the case of Linux, lawyers are currently wrangling over this very question.) Nevertheless, the essence-of-Unix has remained quite stable through all the transformations.

Roots

When Unix was young, computers were scarce and expensive, which meant that each machine had many users. Dozens of terminals would be wired to a single CPU, and it was the function of time-sharing software like Unix to create the illusion that you had a computer of your own. Part of this magic trick lay in arranging for the processor to switch tasks so rapidly that it could run snippets of your program in between other users' keystrokes. Another aspect was walling off each user in a separate compartment, so that no one could trespass on anyone else's space, either deliberately or inadvertently.

Computers are no longer scarce. In many offices and even in some households, they outnumber people. The challenge today is not dividing a single CPU among multiple users but helping each individual manage and coordinate the resources of multiple machines. Where is the latest version of that document you were looking for—on the file server, on your laptop, on the PC at home? These days it could even be on your keychain. A time-sharing operating system is not the obvious solution to such problems. And yet Unix and its offspring have adapted themselves surprisingly well to this new environment.

Even on a computer with just a single user, the task-switching mechanism remains vitally important: It allows you to keep dozens of programs active at the same time. And the barriers originally meant to isolate users from one another now prevent a rogue program from clobbering other

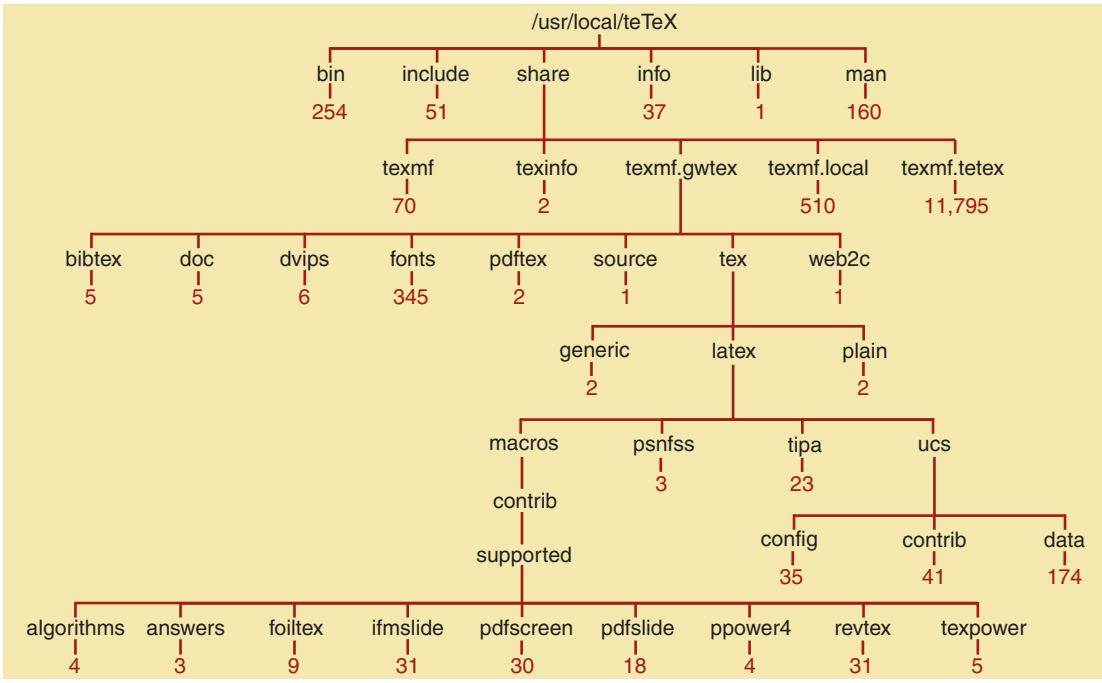

Figure 3. Trees of nested directories, a concept introduced in the first versions of Unix, have become the universal device for organizing computer files. Shown here is part of the tree for an implementation of the TeX typesetting system. Only a few branches are drawn in detail; numbers in color indicate nodes not shown. Altogether there are some 13,700 files and directories. The structure is so elaborate that it has to be built by an automated installer.

software. Admittedly, the multiuser security apparatus can seem a little excessive—a little qwerky—when you're working all by yourself. In the Unix social milieu, each user has a private domain; groups of users can share resources; there's a public space accessible to all; and there's an inner sanctum off limits to everyone but the "superuser," also known as "root." When you try to delete a file and your laptop curtly objects, "You don't have permission to do that," you will probably not run down the hall looking for someone with root privileges.

But the scheme of file permissions, developed for the hub-and-spoke world of time-sharing computers, turns out to work reasonably well in network file systems, which create a decentralized information universe. A network file system frees you to wander from one machine to another within a local network, taking your entire computing environment with you, including your private documents and programs.

Trees

My candidate for the most important single innovation introduced by Unix is the hierarchical file system. I also consider it the feature most desperately in need of a better idea.

In the 1950s, a computer file system was a cabinet full of magnetic tapes, tended by a poor drudge who retrieved them as needed. Disk storage allowed information to be kept on-line without manual intervention, but the early disks were small enough that organizing files was not a serious problem. Nevertheless, Ritchie and Thompson foresaw that a simple, flat list of files would soon become unwieldy. Their solution, a tree of directories nested inside directories, has held up quite well for 35 years. (Incidentally, this phylogeny was recapitulated twice in the ontogeny of microcomputers. The first versions of both MS-DOS and the Macintosh operating system had no nested subdirectories; they were flat file systems. In both cases tree-structured directories were added in the next release.)

The Unix file system has the topology of a rooted tree, a structure made up of linked nodes called parents and children. The root of the tree—which by qwerky tradition is always placed at the top—is a special node that has no parent. Directly below the root are its children, which can have children of their own, and so on to arbitrary depth. Any node can have any number of children (including zero), but every node other than the root has exactly one parent. Because of this single-parent constraint, there is always a unique path from the root to any node of the tree. In other words, you can find any directory or file by starting at the root and following some path from parent to child—and there will be only one such path.

By now both the pleasures and the frustrations of directory trees are familiar to all. When you organize your correspondence, you can do it chronologically, setting up a directory for each year, with nested directories for each month. Or instead you can create a directory for each recipient. Or you can invent topical categories—love letters, crank letters, letters to the editor. The trouble is, any one such scheme precludes all the others. If you arrange the files topically, you can't also keep them in chronological sequence. (A

mechanism called a symbolic link can create shortcuts between distant nodes of the tree, but it's not a practical solution to this problem.)

The great advantage of trees as a data structure is efficiency of access. Suppose you are searching for a specific document among N files. Going from file to file through a flat, unstructured list, the effort required is proportional to N. With a tree of directories, the effort is reduced to the logarithm of N—a tremendous gain. But for *some* value of N, even log N grows unreasonably large.

I recently installed a new implementation of the TeX typesetting system (another magnificently qwerky artifact, although it does not go back quite as far as 1969). The system includes more than 13,000 files; a small part of the tree is shown in Figure 3. Note that for the program to work, it is not enough that all the files be present; they must also be in the right places. The hard work of creating this structure was done by an automated installer that goes out over the Internet, finds the necessary components, and puts them where they belong. I could never have constructed it by hand. If it breaks, I have no idea how to fix it.

I take my own helplessness in this situation as a sign that trees may be nearing their practical limit as an organizational device. For a glimpse of what could lie ahead, look at the third notable technology among the 1969 alumni—the Internet, and specifically the World Wide Web. The topology of the Web is uncannily like that of a Unix file system. Internet domain names take the form of a tree (or rather a small copse of trees, since the top-level domains .com, .org and so on are independent roots). Hyperlinks between Web pages are equivalent to symbolic links to Unix files. The only difference is that the Web is several orders of magnitude larger. So as personal file systems continue to grow, perhaps we will manage them with the same kinds of tools we now use for the Web. Indeed, it is widely assumed that the Web browser will be the model for the next generation of operating system. This is a prospect I do not find reassuring. The Web is a wonderful resource, but few of us would view it as a reliable storage-and-retrieval medium, where you can put something in and count on getting it out again. I'd say it's more like a fishing hole, where you throw your hook in and hope that something bites it. Don't be surprised if you see the error message "Go fish!"

Qwerks to Come

The premise of this column is a bit of fraud. It's too easy to look back 35 years and pick out a few ideas that proved to be winners; the real trick is to name the present-day technologies that will emerge as qwerks of the future. I'm not even sure about the further evolution of the three choices discussed here. In one form or another the Internet will surely survive, and it's a fair bet there will be microprocessors that recognize *x*86 instructions for some time to come. But what will systems software look like in 10 years, or 35?

A number of visionary proposals are on the record already. Jef Raskin, the first architect of the Macintosh, offers the idea of a "zooming space" to replace operating systems, Web browsers and various other kinds of programs. The metaphor is geographic: Documents are spread out over a landscape, and the user soars above them, zooming in to see details and zooming out for an overview. It's an appealing (if slightly dizzying) notion, but I wonder whether it will scale any better than nested trees of directories.

David Gelernter of Yale argues that the key organizing principle should be temporal rather than spatial. His "lifestreams" model arranges documents in sequences and subsequences, with various facilities for browsing and searching. Underneath this interface is a database called tuple space. It's worth noting that both Raskin and Gelernter focus on the problem of organizing *documents*—texts, music, pictures, or more generally what nowadays tends to be called "content." But the original purpose of operating systems was mainly to organize the software that runs the computer itself, and this still needs to be done. Can a zooming space or a lifestream help me manage that gargantuan TeX tree?

The most radical proposal comes from Donald A. Norman, another interface expert who was once at Apple. He solves the complexity problem at a stroke: If computers are too hard to use, just get rid of them! They can be replaced by more-specialized "information appliances"—one device to send e-mail, one to balance your checkbook, one to do tax returns, etc. I don't trust myself to gauge the plausibility of this idea because I find it so thoroughly uncongenial. I want to know which information appliance replaces the computer that I *compute* with.

As for my own predictions, I'm going to paraphrase an old joke that used to be told about the Fortran programming language. I don't know what the operating system of the future will look like. All I know is that it will be called Unix.

Bibliography

Gelernter, David. 1998. *Machine Beauty: Elegance and the Heart of Technology*. New York: Basic Books.

Halfhill, Tom R. 1995. Intel's P6. *Byte* 20(4):42–58.

Intel Corporation. 2003. *IA-32® Architecture Software Developer's Manual. Volume 1: Basic Architecture*. Mt. Prospect, Ill.: Intel Corporation. http://www.intel.com/design/pentium4/manuals/

Norman, Donald A. 1998. *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances are the Solution*. Cambridge, Mass.: MIT Press.

Polsson, Ken. 2003. Chronology of personal computers. http://www.islandnet.com/~kpolsson/comphist/

Raskin, Jef. 2000. *The Humane Interface: New Directions for Designing Interactive Systems*. Reading, Mass.: Addison-Wesley.

Ritchie, Dennis M., and Ken Thompson. 1974. The UNIX™ time-sharing system. *Communications of the ACM* 17:365–375.

Ritchie, Dennis M. 1984. The evolution of the Unix time-sharing system. *Microsystems* 5(10):36–48.