DOI:10.1145/1995376.1995391

Article development led by acmqueue
queue.acm.org

Did Ken, Dennis, and Brian choose wrong with NUL-terminated text strings?

BY POUL-HENNING KAMP

The Most Expensive One-Byte Mistake

INFORMATION TECHNOLOGY (IT) BOTH drives and implements the modern Western-style economy. Thus, we regularly see headlines about staggeringly large amounts of money connected with IT mistakes. Which IT or CS decision has resulted in the most expensive mistake?

Not long ago, a fair number of pundits were doing a lot of hand waving about the financial implications of Sony's troubles with its PlayStation Network, but an event like that does not count here. In my school days, I talked with an inspector from *The Guinness Book of World Records* who explained that for something to be "a true record," it could not be a mere accident; there had to be direct causation starting with human intent (such as, we stuffed 26 high-school students into our music teacher's Volkswagon Beetle and closed the doors).

Sony (probably) did not *intend* to see how big a mess it could make with the least attention to security, so this and other such examples of false economy will not qualify. Another candidate could be IBM's choice of Bill Gates over Gary Kildall to supply the operating system for its personal computer. The damage from this decision is still accumulating at breakneck speed, with StuxNet and the OOXML perversion of the ISO standardization process being exemplary bookends for how far and wide the damage spreads. But that was not really an IT or CS decision. It was a business decision that, as far as history has been able to uncover, centered on Kildall's decision not to accept IBM's nondisclosure demands.

A better example would be the decision for MS-DOS to invent its own directory/filename separator, using the backslash (\) rather than the forward slash (/) that Unix used or the period that DEC used in its operating systems. Apart from the actual damage being relatively modest, however, this does not qualify as a good example either because it was not a real decision selecting a true preference. IBM had decided to use the slash for command flags, eliminating Unix as a precedent, and the period was used between filename and filename extension, making it impossible to follow DEC's example.

Space exploration history offers a pool of well-publicized and expensive mistakes, but interestingly, I did not find any valid candidates there. Fortran syntax errors and space shuttle computer synchronization mistakes do not qualify for lack of intent. Running one part of a project in imperial units and the other in metric is a "random act of management" that has nothing to do with CS or IT.

The best candidate I have been able to come up with is the C/Unix/Posix use of NUL-terminated text strings. The choice was really simple: Should the C language represent strings as an address + length tuple or just as the

address with a magic character (NUL) marking the end? This is a decision that the dynamic trio of Ken Thompson, Dennis Ritchie, and Brian Kernighan must have made one day in the early 1970s, and they had full freedom to choose either way. I have not found any record of the decision, which I admit is a weak point in its candidacy: I do not have proof that it was a conscious decision.

As far as I can determine from my research, however, the `address + length` format was preferred by the majority of programming languages at the time, whereas the `address + magic _ marker` format was used mostly in assembly programs. As the C language was a development from assembly to a portable high-level language, I have a difficult time believing Ken, Dennis, and Brian gave it no thought.

Using an `address + length` format would cost one more byte of overhead than an `address + magic _ marker` format, and their PDP computer had limited core memory. In other words, this could have been a perfectly typical and rational IT or CS decision, like the many similar decisions we all make every day; but this one had quite atypical economic consequences.

Hardware development costs. Initially, Unix had little impact on hardware and instruction set design. The CPUs that offered string manipulation instructions—for example, Z-80 and DEC VAX—did so in terms of the far more widespread `adr+len` model. Once Unix and C gained traction, however, the terminated string appeared on the radar as an optimization target, and CPU designers started to add instructions to deal with them. One example is the Logical String Assist instructions IBM added to the ES/9000 520-based processors in 1992.[1]

Adding instructions to a CPU is not cheap, and it happens only when there are tangible and quantifiable monetary reasons to do so.

Performance costs. IBM added instructions to operate on NUL-terminated strings because its customers spent expensive CPU cycles handling such strings. That bit of information, however, does not tell us if fewer CPU cycles would have been required if a `ptr+len` format had been used.

Thinking a bit about virtual memory (VM) systems settles that question for us. Optimizing the movement of a known-length string of bytes can take advantage of the full width of memory buses and cache lines, without ever

ILLUSTRATION BY GARY NEILL

touching a memory location that is not part of the source or destination string.

One example is FreeBSD's `libc`, where the `bcopy(3)/memcpy(3)` implementation will move as much data as possible in chunks of "unsigned long," typically 32 bits or 64 bits, and then "mop up any trailing bytes" as the comment describes it, with byte-wide operations.[2]

If the source string is NUL terminated, however, attempting to access it in units larger than bytes risks attempting to read characters after the NUL. If the NUL character is the last byte of a VM page and the next VM page is not defined, this would cause the process to die from an unwarranted "page not present" fault.

Of course, it is possible to write code to detect that corner case before engaging the optimized code path, but this adds a relatively high fixed cost to all string moves just to catch this unlikely corner case—not a profitable trade-off by any means.

If we have out-of-band knowledge of the strings, things are different.

Compiler development cost. One thing a compiler often knows about a string is its length, particularly if it is a constant string. This allows the compiler to emit a call to the faster `memcpy(3)` even though the programmer used `strcpy(3)` in the source code.

Deeper code inspection by the compiler allows more advanced optimizations, some of them very clever, but only if somebody has written the code for the compiler to do it. The development of compiler optimizations has historically been neither easy nor cheap, but obviously Apple is hoping this will change with Low-level Virtual Machine (LLVM), where optimizers seem to come *en gros*.

The downside of heavy-duty compiler optimization—in particular, optimizations that take holistic views of the source code and rearrange it in large-scale operations—is that the programmer must be really careful that the source code specifies his or her complete intention precisely. A programmer who worked with the compilers on the Convex C3800 series supercomputers related his experience as "having to program as if the compiler was my ex-wife's lawyer."

Security costs. Even if your compiler does not have hostile intent, source code should be written to hold up to attack, and the NUL-terminated string has a dismal record in this respect. Utter security disasters such as `gets(3)`, which "assume the buffer will be large enough," are a problem "we have relatively under control."[3]

Getting it under control, however, takes additions to compilers that would complain if the `gets(3)` function were called. Despite 15 years of attention, over- and underrunning string buffers is still a preferred attack vector for criminals, and far too often it pays off.

Mitigation of these risks has been added at all levels. Long-missed no-execute bits have been added to CPUs' memory management hardware; operating systems and compilers have added address-space randomization, often at high costs of performance; and static and dynamic analyses of programs have soaked up countless hours, trying to find out if the byzantine diagnostics were real bugs or clever programming.

Yet, absolutely nobody would be surprised if Sony's troubles were revealed to start with a buffer overflow or false NUL-termination assumption.

Slashdot Sensation Prevention Section

We learn from our mistakes, so let me say for the record, before somebody comes up with a catchy but totally misleading Internet headline for this article, that there is absolutely no way Ken, Dennis, and Brian could have foreseen the full consequences of their choice some 30 years ago, and they disclaimed all warranties back then. For all I know, it took at least 15 years before anybody realized why this subtle decision was a bad idea, and few, if any, of my own IT decisions have stood up that long.

In other words, Ken, Dennis, and Brian did the right thing.

But That Doesn't Solve the Problem

To a lot of people, C is a dead language, and ${lang} is the language of the future, for ever-changing transient values of ${lang}. The reality of the situation is that all other languages today directly or indirectly sit on top of Posix API and the NUL-terminated string of C.

When your Java, Python, Ruby, or Haskell program opens a file, its runtime environment passes the filename as a NUL-terminated string to `open(3)`, and when it resolves cacm.acm.org to an IP number, it passes the host name as a NUL-terminated string to `getaddrinfo(3)`. As long as you keep doing that, you retain all the advantages when running your programs on a PDP/11, and all of the disadvantages if you run them on anything else.

I could write a straw-man API proposal here, suggest representations, operations, and error-handling strategies, and I am quite certain it would be a perfectly good waste of a nice afternoon. Experience shows that such proposals go nowhere because the backward compatibility with the PDP/11 and the finite number of programs written are much more important than the ability to write the potentially infinite number of programs in the future in an efficient and secure way.

Thus, the costs of the Ken, Dennis, and Brian decision will keep accumulating, like the dust that over the centuries has almost buried the monuments of ancient Rome. C

Related articles on queue.acm.org

Massively Multiplayer Middleware
Michi Henning
http://queue.acm.org/detail.cfm?id=971591

The Seven Deadly Sins of Linux Security
Bob Toxen
http://queue.acm.org/detail.cfm?id=1255423

B.Y.O.C. (1,342 Times and Counting)
Poul-Henning Kamp
http://queue.acm.org/detail.cfm?id=1944489

References

1. Computer Business Review. Partitioning and Escon enhancements for top-end ES/9000s (1992); http://www.cbronline.com/news/ibm_announcements_71.
2. ViewVC. Contents of /head/lib/libc/string/bcopy.c (2007); http://svnweb.freebsd.org/base/head/lib/libc/string/bcopy.c?view=markup.
3. Wikipedia. Lifeboat sketch (2011); http://en.wikipedia.org/wiki/Lifeboat_sketch.

Poul-Henning Kamp (phk@FreeBSD.org) has programmed computers for 26 years and is the inspiration behind bikeshed.org. His software has been widely adopted as "under the hood" building blocks in both open source and commercial products. His most recent project is the Varnish HTTP accelerator, which is used to speed up large Web sites such as Facebook.