the Availability Digest

www.availabilitydigest.com
@availabilitydig

Not Only the Y2038 Problem – There's a Y2028 Problem

May 2017

In our article "Future Dates Spell Problems for IT" in the March, 2017, issue of the *Availability Digest*,[1] we pointed out that many systems might crash on January 19, 2038, due to the overflow of 32-bit date/time fields. Though most applications should be corrected by then to avoid this problem, there are old legacy applications still in use that are not easily modifiable. For many, the source code is lost and the original programmers have all moved on. These applications are at risk for this fault.

For many systems, such a catastrophe might occur a decade earlier, in the year 2028. But this will occur for a completely different reason - a Y2K temporary fix that has become all but temporary.

Y2K was a big problem for many programs. Written years ago, these applications used a two-digit year field to save memory space, which was at a premium when these programs were written. The year '1967' was stored as '67.' But as the year 2000 approached, the year 2000 would be interpreted as the year 1900. All later years in the 21st century would be treated as 20th century years.

Massive efforts were launched to modify applications to move from a two-digit year field to a four-digit year field. However, some clever individuals came up with another technique. Since the calendar repeats itself every 28 years, it was only necessary to roll the calendar back 28 years.[2] That is seven years to get the week days synchronized, times four to account for the leap year. The year 2000 became the year 1972. The year 2017 became the year 1989. The application then only had to be modified to add 28 to the year. For instance, if the year was stored as '89,' the modified application would interpret this as 1989 + 28 = 2017.

This was a much easier change to the applications in many cases. For instance, data stored in the database could still use two-digit dates. If a full Y2K correction were implemented, all database structures that stored a date would have to be modified to use a four-digit year rather than a two-digit year.

Unfortunately, this algorithm breaks down in the year 2028. The year 2027 is stored as the year 1999, or as '99' in the two digit format. The next year is stored as '00,' which the application interprets as the year 1900. Adding 28 gives the year 1928 instead of the year 2028. At this point, application programs will fail. Imagine what happens if you try to calculate interest from the year 2027 to the year 1928? In fact, CNBC reported that the Congressional Budget Office's computer program crashed when it reached the year 2027 as it calculated forward government spending.

Clearly, these applications will have to be corrected before they start using dates beyond 2027. They will have to finally be modified to use a four-digit year. Using the Year 2028 fix bought companies a 28-year reprieve, but time will catch up to them.

---

[1] Future Dates Spell Problems for IT, *Availability Digest*; March 2017.
http://www.availabilitydigest.com/public_articles/1203/dates.pdf

[2] Y2K28 Problem: All Computers Will Crash in 16 Years, *Godlike Productions*; July 28, 2012.
http://www.godlikeproductions.com/forum1/message1939123/pg1

© 2017 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com

Many people are saying that this won't be a big problem since most computers in use today will have been replaced by 2028. However, the Y2028 problem is not a computer problem. It is an application problem, and the applications must be retired or fixed. For those unmodified applications that are still around by the year 2028 (or earlier if the applications are performing time-forward computations), beware the Y2028 bug. It could deliver a painful bite.

Acknowledgement

Thanks to our subscriber, Alan Dick, for pointing us to this issue.

© 2017 Sombers Associates, Inc., and W. H. Highleyman
www.availabilitydigest.com
For discussion, contact editor@availabilitydigest.com