Year 2038 problem

The **Year 2038 problem** (also called **Y2038**, **Epochalypse**,[1][2] **Y2k38**, or **Unix Y2K**) relates to representing time in many digital systems as the number of seconds passed since 00:00:00 UTC on 1 January 1970 and storing it as a signed 32-bit integer. Such implementations cannot encode times after 03:14:07 UTC on 19 January 2038. Similar to the Y2K problem, the Year 2038 problem is caused by insufficient capacity used to represent time.

One of the possible manifestations of the bug on a specific machine: the date could reset at 03:14:08 UTC on 19 January 2038.

Contents

Cause

The latest time since 1 January 1970 that can be stored using a signed 32-bit integer is 03:14:07 on Tuesday, 19 January 2038 (2^{31}-1 = 2,147,483,647 seconds after 1 January 1970).[3]

Programs that attempt to increment the time beyond this date will cause the value to be stored internally as a negative number, which these systems will interpret as having occurred at 20:45:52 on Friday, 13 December 1901 (2,147,483,648 seconds before 1 January 1970) rather than 19 January 2038. This is caused by integer overflow, during which the counter runs out of usable binary digits or bits, and flips the sign bit instead. This reports a maximally negative number, and continues to count *up*, towards zero, and then up through the positive integers again. Resulting erroneous calculations on such systems are likely to cause problems for users and other reliant parties.

Early problems

In May 2006, reports surfaced of an early manifestation of the Y2038 problem in the AOLserver software. The software was designed with a kludge to handle a database request that should "never" time out. Rather than specifically handling this special case, the initial design simply specified an arbitrary time-out date in the future. The default configuration for the server specified that the request should time out after one billion seconds. One billion seconds (approximately 32 years) after 01:27:28 UTC on 13 May 2006 is beyond the 2038 cutoff date. Thus, after this time, the time-out calculation overflowed and returned a date that was actually in the past, causing the software to crash. When the problem was discovered, AOLServer operators had to edit the configuration file and set the time-out to a lower value.[4][5]

Players of games or apps which are programmed to impose waiting periods[6] are running into this problem when the players try to bypass the waiting period by setting the date on their devices to a date past 19 January 2038, but are unable to do so, since a 32-bit Unix time format is being used.

Vulnerable systems

Embedded systems that use dates for either computation or diagnostic logging are most likely to be affected by the 2038 problem.[7]

Many transportation systems from flight to automobiles use embedded systems extensively. In automotive systems, this may include anti-lock braking system (ABS), electronic stability control (ESC/ESP), traction control (TCS) and automatic four-wheel drive; aircraft may use inertial guidance systems and GPS receivers.[note 1] However, this does not imply that all these systems will suffer from the Y2038 problem, since many such systems do not require access to dates. For those that do, those systems which only track the difference between times/dates and not absolute times/dates will, by the nature of the calculation, not experience a major problem. This is the case for automotive diagnostics based on legislated standards such as CARB (California Air Resources Board).[8]

Another major use of embedded systems is in communications devices, including cell phones and Internet appliances (routers, wireless access points, IP cameras, etc.) which rely on storing an accurate time and date and are increasingly based on UNIX-like operating systems. For example, the Y2038 problem makes some devices running 32-bit Android crash and not restart when the time is changed to that date.[9]

Despite the modern 18–24 month generational update in computer systems technology, embedded systems are designed to last the lifetime of the machine in which they are a component. It is conceivable that some of these systems may still be in use in 2038. It may be impractical or, in some cases, impossible to upgrade the software running these systems, ultimately requiring replacement if the 32-bit limitations are to be corrected.

MySQL database's built-in functions like `UNIX_TIMESTAMP()` will return 0 after 03:14:07 UTC on 19 January 2038.[10]

Early Mac OS X versions[11] are susceptible to the Year 2038 problem.

Data structures with time problems

Many data structures in use today have 32-bit time representations embedded into their structure. A full list of these data structures is virtually impossible to derive but there are well-known data structures that have the Unix time problem:

- file systems (many file systems use only 32 bits to represent times in inodes)
- binary file formats (that use 32-bit time fields)
- databases (that have 32-bit time fields)

- database query languages, like SQL that have `UNIX_TIMESTAMP()`-like commands

Examples of systems using data structures that may contain 32-bit time representations include:

- embedded factory, refinery control and monitoring subsystems
- assorted medical devices
- assorted military devices

Any system making use of data structures containing 32-bit time representations will present risk. The degree of risk is dependent on the mode of failure.

Possible solutions

There is no universal solution for the Year 2038 problem. For example, in the C language, any change to the definition of the `time_t` data type would result in code-compatibility problems in any application in which date and time representations are dependent on the nature of the signed 32-bit `time_t` integer. For example, changing `time_t` to an unsigned 32-bit integer, which would extend the range to 2106 (specifically, 06:28:15 UTC on Sunday, 7 February 2106), would adversely affect programs that store, retrieve, or manipulate dates prior to 1970, as such dates are represented by negative numbers. Increasing the size of the `time_t` type to 64 bits in an existing system would cause incompatible changes to the layout of structures and the binary interface of functions.

Most operating systems designed to run on 64-bit hardware already use signed 64-bit `time_t` integers. Using a signed 64-bit value introduces a new wraparound date that is over twenty times greater than the estimated age of the universe: approximately 292 billion years from now. The ability to make computations on dates is limited by the fact that `tm_year` uses a signed 32-bit integer value starting at 1900 for the year. This limits the year to a maximum of 2,147,485,547 (2,147,483,647 + 1900).[12]

FreeBSD uses 64-bit `time_t` for all 32-bit and 64-bit architectures except 32-bit i386, which uses signed 32-bit `time_t` instead.[13]

Starting with NetBSD version 6.0 (released in October 2012), the NetBSD operating system uses a 64-bit `time_t` for both 32-bit and 64-bit architectures. Applications that were compiled for an older NetBSD release with 32-bit `time_t` are supported via a binary compatibility layer, but such older applications will still suffer from the Year 2038 problem.[14]

OpenBSD since version 5.5, released in May 2014, also uses a 64-bit `time_t` for both 32-bit and 64-bit architectures. In contrast to NetBSD, there is no binary compatibility layer. Therefore, applications expecting a 32-bit `time_t` and applications using anything different from `time_t` to store time values may break.[15]

Linux originally used a 64-bit `time_t` for 64-bit architectures only; the pure 32-bit ABI was not changed due to backward compatibility.[16] Starting with version 5.6, 64-bit `time_t` is supported on 32-bit architectures, too. This was done primarily for the sake of embedded Linux systems.[17]

The x32 ABI for Linux (which defines an environment for programs with 32-bit addresses but running the processor in 64-bit mode) uses a 64-bit `time_t`. Since it was a new environment, there was no need for special compatibility precautions.[16]

While the native APIs of OpenVMS can support timestamps up to the 31st of July 31086,[18] the C runtime library (CRTL) uses 32-bit integers for `time_t`.[19] As part of Y2K compliance work that was carried out in 1998, the CRTL was modified to use unsigned 32-bit integers to represent time; extending the range of `time_t` up to the 7th of February 2106.[20]

Network File System version 4 has defined its time fields as `struct nfstime4 {int64_t seconds; uint32_t nseconds;}` since December 2000.[21] Values greater than zero for the seconds field denote dates after the 0-hour, January 1, 1970. Values less than zero for the seconds field denote dates before the 0-hour, January 1, 1970. In both cases, the nseconds (nanoseconds) field is to be added to the seconds field for the final time representation.

Alternative proposals have been made (some of which are already in use), such as storing either milliseconds or microseconds since an epoch (typically either 1 January 1970 or 1 January 2000) in a signed 64-bit integer, providing a minimum range of 300,000 years at microsecond resolution.[22][23] In particular, Java's use of 64-bit long integers everywhere to represent time as "milliseconds since 1 January 1970" will work correctly for the next 292 million years. Other proposals for new time representations provide different precisions, ranges, and sizes (almost always wider than 32 bits), as well as solving other related problems, such as the handling of leap seconds. In particular, TAI64[24] is an implementation of the International Atomic Time (TAI) standard, the current international real-time standard for defining a second and frame of reference.

See also

- *Deep Impact* is believed to have been lost at the time its internal clock reached 2^{32} 100-millisecond intervals (one-tenth second) since 2000, on 11 August 2013, 00:38:49 UTC.
- John Titor, self-described time traveler who is sometimes related to the problem
- Time formatting and storage bugs

Notes

1. GPS suffers its own time counter overflow problem known as GPS Week Number Rollover.

References

1. Bergmann, Arnd (6 February 2020). "The end of an Era" (https://www.linaro.org/blog/the-end-of-an-era/). Linaro.
2. Wagenseil, Paul (28 July 2017). "Digital 'Epochalypse' Could Bring World to Grinding Halt" (https://www.tomsguide.com/us/2038-bug-bh2017,news-25551.html). *Tom's Guide*.
3. Diomidis Spinellis (2006). *Code quality: the open source perspective* (https://books.google.com/books?id=vEN-ckcdtCwC&q=292%2C277%2C026%2C596&pg=PA49). Effective software development series in Safari Books Online (illustrated ed.). Adobe Press. p. 49. ISBN 978-0-321-16607-4.
4. "The Future Lies Ahead" (http://substitute.livejournal.com/1430908.html). 28 June 2006. Retrieved 19 November 2006.
5. Weird "memory leak" problem in AOLserver 3.4.2/3.x (http://www.mail-archive.com/aolserver@listserv.aol.com/msg09844.html) 12 May 2006
6. Fahey, Mike (21 January 2013). "Infinite Lives in *Candy Crush Saga* Isn't Cheating, It's Time Travel" (https://kotaku.com/5977630/infinite-lives-in-candy-crush-saga-isnt-cheating-its-time-travel). *Kotaku*.

7. "Is the Year 2038 problem the new Y2K bug?" (https://www.theguardian.com/technology/2014/dec/17/is-the-year-2038-problem-the-new-y2k-bug). *The Guardian*. 17 December 2014. Retrieved 11 October 2018.
8. "ARB Test Methods / Procedures" (http://www.arb.ca.gov/testmeth/testmeth.htm#vehicles). *ARB.ca.gov*. California Air Resources Board.
9. "ZTE Blade running Android 2.2 has 2038 problems" (https://issuetracker.google.com/issues/36928638). Retrieved 20 November 2018.
10. "MySQL Bugs: #12654: 64-bit unix timestamp is not supported in MySQL functions" (https://bugs.mysql.com/bug.php?id=12654). *bugs.mysql.com*.
11. "unix - Is any version of OS X/macOS vulnerable to the Year 2038 problem?" (https://apple.stackexchange.com/questions/252542/is-any-version-of-os-x-macos-vulnerable-to-the-year-2038-problem). *Ask Different*. Retrieved 12 October 2019.
12. Felts, Bob (17 April 2010). "The End of Time" (http://stablecross.com/files/End_Of_Time.html). *Stablecross.com*. Retrieved 19 March 2012.
13. https://www.freebsd.org/cgi/man.cgi?arch
14. "Announcing NetBSD 6.0" (https://www.netbsd.org/releases/formal-6/NetBSD-6.0.html). 17 October 2012. Retrieved 18 January 2016.
15. "OpenBSD 5.5 released (May 1, 2014)" (http://www.openbsd.org/plus55.html). 1 May 2014. Retrieved 18 January 2016.
16. Jonathan Corbet (14 August 2013). "Pondering 2038" (https://lwn.net/Articles/563285/). *LWN.net*. Archived (https://web.archive.org/web/20160304081847/https://lwn.net/Articles/563285/) from the original on 4 March 2016. Retrieved 9 March 2016.
17. "LKML: Arnd Bergmann: [GIT PULL] y2038: core, driver and file system changes" (https://lkml.org/lkml/2020/1/29/355?anz=web). *lkml.org*. Retrieved 30 January 2020.
18. "Why is Wednesday, November 17, 1858 the base time for OpenVMS (VAX VMS)?" (https://www.slac.stanford.edu/~rkj/crazytime.txt). *Stanford University*. 24 July 1997. Archived (https://web.archive.org/web/19970724202734/https://www.slac.stanford.edu/~rkj/crazytime.txt) from the original on 24 July 1997. Retrieved 8 January 2020.
19. "VSI C Run-Time Library Reference Manual for OpenVMS Systems" (https://vmssoftware.com/docs/VSI_CRTL_REF.pdf) (PDF). VSI. November 2020. Retrieved 17 April 2021.
20. "OpenVMS and the year 2038" (https://www.zx.net.nz/mirror/h71000.www7.hp.com/2038.html). HP. Retrieved 17 April 2021.
21. Haynes, Thomas; Noveck, David, eds. (March 2015). "Structured Data Types" (https://tools.ietf.org/html/rfc7530#section-2.2). *Network File System (NFS) Version 4 Protocol* (https://tools.ietf.org/html/rfc7530). sec. 2.2. doi:10.17487/RFC7530 (https://doi.org/10.17487%2FRFC7530). RFC 7530 (https://tools.ietf.org/html/rfc7530).
22. "Ununnium Time" (https://web.archive.org/web/20060408161959/http://ununnium.org/articles/uuutime). Archived from the original (http://ununnium.org/articles/uuutime) on 8 April 2006. Retrieved 19 November 2006.
23. Sun Microsystems. "Java API documentation for System.currentTimeMillis()" (https://docs.oracle.com/javase/9/docs/api/java/lang/System.html#currentTimeMillis--). Retrieved 29 September 2017.
24. "TAI64" (http://cr.yp.to/libtai/tai64.html).

External links

- Y2038 Proofness Design (https://sourceware.org/glibc/wiki/Y2038ProofnessDesign) glibc Wiki
- Entry in How Stuff Works (https://computer.howstuffworks.com/question75.htm)
- The Project 2038 Frequently Asked Questions (http://maul.deepsky.com/%7Emerovech/2038.html)

- *Critical and Significant Dates* 2038 (https://people.cs.nctu.edu.tw/~tsaiwn/sisc/runtime_error_200_div_by_0/www.merlyn.demon.co.uk/critdate.htm#Y2038)
- A 2038-safe replacement for time.h on 32 bit systems (https://github.com/evalEmpire/y2038/wiki)
- Baraniuk, Chris (5 May 2015). "The number glitch that can lead to catastrophe" (https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster). *BBC Future*.
- Clewett, James. "2,147,483,647 – The End of Time [Unix]" (https://web.archive.org/web/20170522031114/http://www.numberphile.com/videos/unix_time.html). *Numberphile*. Brady Haran. Archived from the original (http://www.numberphile.com/videos/unix_time.html) on 22 May 2017. Retrieved 7 April 2013.
