IBM PC Real Time Clock should run in UT

Markus Kuhn, Computer Laboratory, University of Cambridge

A plea to the developers in charge of the real-time clock driver in Microsoft operating systems.

Background

What is Universal Time? Universal Time (UT), also known as Greenwich Mean Time (GMT), is the astronomical local time on the prime meridian. Its atomic-clock representation, known as Coordinated Universal Time (UTC) and defined in ITU-R Recommendation TF.460-4, is the official international reference time scale, the basis all time announcement services and most national definitions of time. Universal Time is not affected by any national Daylight Saving Times. It has long become the global reference time in distributed computer systems and many other applications.

What is the PC's Real Time Clock? Every IBM PC compatible computer features a battery-backed "Real Time Clock (RTC)" chip. This contains a 32768 Hz crystal oscillator that drives a date/time counter. It also contains 114 bytes of battery-backed CMOS RAM that are reserved for BIOS use. The chip typically used is compatible to the Dallas Semiconductor DS1287 and Motorola MC146818B RTCs.

How does MS-DOS relate to this? The original IBM PC was released with the first Microsoft DOS and Basic versions in the early 1980s. At that time, the engineers involved did not consider modern issues such as networking across time zones, mobile computing, automatic clock synchronisation, automatic daylight saving time updates, 24/7 database server operation, dual-boot installations, virtual-machine servers, etc. Microsoft and IBM established at the time the (now highly problematic) convention to keep the RTC adjusted to whatever the PC's user considers her local time zone. MS-DOS never had any notion of Universal Time or any time zone other than "local time". The question of time zones only emerged when MS-DOS C compilers began to implement the standard C API function *gmtime()*, which applications can use to query Universal Time. They followed a practice now documented in IEEE POSIX.1 Section 8.3, namely interpreting the environment variable TZ as a description of local time relative to Universal Time, in order to convert the local time provided by MS-DOS into Universal Time.

This never worked properly, especially not near the repeated hour after the end of a daylight-saving-time period. Since an entire hour is repeated once every year, there is no unambiguous way for a function to convert local time into Universal Time, leading to wild jumps in the output of *gmtime()* under MS-DOS each fall.

MS-DOS never was designed for running mission-critical 24/7 servers reliably, so this remained a minor concern. For those who cared, replacement RTC drivers (CLOCK.SYS) appeared eventually for MS-DOS, such as clk360rs.zip, which allowed users to run the RTC directly in UTC and which determined the local time based on a configured TZ string when the system boots.

The POSIX platform and C programming language had been developed from early on for use in global networks across many time zones. Most POSIX systems keep their hardware real-time clock in Universal Time, and that is also the only time zone that the kernel of a standard POSIX system is aware of (e.g. in file modification time stamps). Conversion to any displayed local time zones is performed on POSIX systems by the C run-time library, either based on the TZ environment variable or time zone information stored in configuration files.

Eventually, the Microsoft Windows API also introduced a notion of time zones and application access to Universal Time, but **even today all Microsoft operating systems still continue with the dangerous old MS-DOS practice of running the RTC in local time**. Even the current Microsoft Windows 7 and the server versions do that by default!

The problem

The continued use of local time in the RTC by Microsoft operating systems causes various problems:

- Universal Time can be unambiguously converted into a local time. The converse is not true, that is a local time cannot reliably be converted back into Universal time. This is due to the *Summer Time* or *Daylight Savings Time* offset periods implemented in many countries. At the end of that offset period, local-time clocks have to be turned back by usually one hour, therefore a 60 minute period on the local-time scale is repeated. There are two possible corresponding hours on the Universal Time scale for the repeated local-time perios, which makes it impossible to unambiguously determine Universal Time during that hour. This obviously makes local time a bad choice for the time format maintained in an independently-running hardware time base. If a computer is booted during the hour after DST ends, it cannot determine UT reliably.
- Daylight Savings Time makes it necessary to readjust the RTC twice per year. However, there exists currently no convention to label in the CMOS RAM, whether that adjustment has already been performed or not. As a result, the operating systems can get confused and will apply the correction multiple times. One possible workaround is to record somewhere on the harddisk, whether the DST change has already been performed this year or not, or what the current offset to UT is, and recent versions of Windows appear to do that in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation \ActiveTimeBias. However this fails for users who have a requirement to run several operating system versions in different harddisk partitions on the same computer. They share the same RTC but lack access to each others configuration files. Similar problems occur when booting from a USB sick or using some PC-in-PC emulator.
- Thanks to virtual-machine based server consolidation, it is now common practice to run many operating-system instances on the same computer. All these operating systems would have to communicate with each other somehow, whether the DST change in the RTC has already happened or not and who is responsible for it. It is much easier to keep the RTC running in Universal Time, which eliminates the problem entirely.
- The numerous past malfunctions of Microsoft operating systems caused by keeping local time in the RTC and trying to cleverly perform the RTC adjustment semi-automatically have been repeatedly the subject of concerned public discussion: RISKS 16.54.1 RISKS 18.96.3 RISKS 19.11.16, RISKS 19.12.14 RISKS 19.43.13, RISKS 19.43.14, RISKS 22.34.3.

The solution

I would like to urge Microsoft: **please follow the time-proven POSIX practice of running the RTC in Universal Time**. Ideally, this should become the standard configuration for the Microsoft platform. At least optional support would be extremely desirable. This way all described problems are solved easily:

- Universal time can unambiguously be converted into any local time, so no undefined rare situations occur during the hour after DST ended.
- Users will not have to be bothered any more with pop-up queries that ask whether the RTC clock should be updated. With a Universal Time RTC, Windows machines can now reliably update their local time at exactly the right time without any delay and the need for manual user intervention, which is especially critical for reliable unsupervised operation in embedded systems, data centers, etc.
- Problems of multiple DST switches in the case of multiple installed operating system versions are avoided and no communication is necessary between different operating system versions to coordinate any RTC DST adjustment.
- Problems of multiple DST switches in the case of reboots or power-ups during the first hour after DST are avoided, which removes a tedious-to-test special case for safety checks of system installations.
- An RTC running in Universal Time does not have to be adjusted when a mobile computer is moved to a new time zone.
- Windows currently offers an option whether the DST update should be performed automatically or not. This option becomes unnecessary once the RTC runs in Universal Time, because the risks associated with an automatic DST update of the RTC will be gone.
- Reliable guaranteed monotonic UT is essential for numerous applications such as distributed databases, automated updating in mirrored file systems, dependency checks in source code management tools, security audit logs, and many other applications. If the operating systems refrain from performing DST updates on the hardware clock, its monotonicity is far easier to

guarantee.

- If the hardware clock runs in UTC, there is no need any more to keep timezone-related functionality in the Windows kernel, as the need to perform a local-time to UTC conversion during boot or after hibernation falls away. Simpler is better!
- In dual-boot Windows/POSIX machines, the POSIX installation with its normally clean and consistent UT-only time handling will not be forced any more to maintain the risky practice of running the RTC in local time, which is frequently done today just for the sake of boot compatibility with a parallel Windows installation.

It seems to me a really rather simple change to add to the Windows RTC driver in every Microsoft operating system a little configuration option that determines whether the time stored in the RTC is Universal Time or local time. I very much hope, Microsoft will consider implementing that at least as an option. Many discussions of this proposal on the comp.protocols.time.ntp newsgroup and the tz mailing list showed enthusiastic support from other computer time keeping experts for this simple proposal.

Initially, it should perhaps only be an option for administrators of 24/7 servers in data centers and users multi-OS-version machines, but eventually, there is really no reason why the Universal Time RTC shouldn't become the standard for *every* new or updated Microsoft installation. After all, correct and reliable knowledge of Universal Time by applications is required by numerous Internet standard protocols (e.g., RFC 822). The time and time zone have to be configured correctly by the user anyway during the installation of a new operating system, which is a good opportunity to set the RTC to Universal Time.

In particular, **no changes are necessary in existing PC hardware and BIOS firmware** in order to run the RTC in UT. This is entirely up to the operating system. The BIOS is already completely ignorant about what time zone its RTC runs in, and it can continue to remain so when the operating system decides that this timezone always be UTC.

The perhaps only change BIOS vendors might eventually want to make is in the documentation of the BIOS configuration menus, which should then encourage users to enter Universal Time and not some local time. But non-technical users are today no longer expected anyway to use the BIOS configuration menus, and should certainly not be encouraged to do so in the future. Perhaps, access to the battery clock from BIOS menus should be moved generally into "for expert users only" sections.

Current support status in Windows

While browsing through the Windows 2000 SP2 kernel binary with the "strings" tool, I noted a UTF-16 encoded string "RealTimeIsUniversal" (NTOSKRNL.EXE:bbd4, NTKRNLPA.EXE:9304). It turned out that Windows NT tests a long forgotten undocumented registry entry

```
HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation\RealTimeIsUniversal
```

That tells the kernel that the RTC runs in UT (type REG_DWORD, values 0 for local time or 1 for Universal Time). When you set *RealTimeIsUniversal*=1, the kernel initialization code that reads the CMOS clock will skip over the section that translates local time into Universal Time and will take the CMOS time directly as the Universal Time kept by the NT kernel clock.

2001-07-09: I got a reply from someone in Microsoft's Base Kernel Team who got interested in *RealTimeIsUniversal* and they had a look at the relevant parts of the NT kernel source code. The *RealTimeIsUniversal* flag is there (a leftover from the days when NT still ran on RISC machines with UTC RTCs), but its implementation seems now incomplete and it is currently not covered by Microsoft's documentation and regression test suite, therefore using it is not recommended at this time. A couple of potential *RealTimeIsUniversal* bugs have been identified over the past few days, there might be more. For instance, the kernel debugger assumes that the CMOS time is local time and will get the time wrong when *RealTimeIsUniversal*=1. There might be a similar problem in the code that resumes processing after the CPU was suspended or in the code that calculates DST change times. I hope they will look into fixing these problems, but they haven't made any promise yet that *RealTimeIsUniversal*=1 will be officially supported. In any case, it is unfortunately too late at this stage for a fix to get into the forthcoming Windows XP release. Perhaps *RealTimeIsUniversal*=1 can be established as the default for new platforms such as IA64 where there is no DOS-compatibility requirement, and then it would be fully supported again?

2004-09-02: A blog post by Windows developer Raymond Chen suggests that (*a*) the Windows development team really hasn't grasped the magnitude of problems caused by day-light switching the RTC (especially in dual-boot environments), and that (*b*) Microsoft might be more inclined to move to UTC if BIOS manufacturers simply removed the RTC clock display from their setup menus, because some Microsoft folks appear to think that displaying UTC in the built-in BIOS config menu (next to other arcane settings such as the type of floppy drive used) is still way too scary for mere mortals. (But none of what he says justifies not even fixing the *RealTimeIsUniversal*=1 support, which could simply be used only by UTC-aware customers who know exactly what they are doing.)

2006-07-04: Various Microsoft Windows Vista beta testers have told me that this next-generation operating system still is not capable of running the CMOS clock in UTC.

2006-11-02: Microsoft and Novell announce broad collaboration on Windows and Linux interoperability and support – full support for *RealTimeIsUniversal*=1 would seem like something that ought to be high on their list ...

2008-10-31: Hurray! Someone from Microsoft's Core Operating Systems Division hints in an email to me that both Vista SP2 and Windows 7 will fix the problems in the *RealTimeIsUniversal*=1 support that have made running the CMOS clock in UTC so far not practical with Windows (i.e., the time was wrong after resuming a suspended/hibernating Windows). He warns though that this improved support for UTC in the CMOS clock may not immediately be widely documented and details may change in the future. I'm also being told that when *RealTimeIsUniversal*=1, then Windows Vista SP2 and 7 will no longer update the CMOS clock, so you may want to run an NTP client that does that in the other operating system. This should make life substantially easier for people who use Windows in a dual-boot environment (e.g., alongside MacOS).

To set this option invoke "cmd" as Administrator, then type

```
reg add HKLM\System\CurrentControlSet\Control\TimeZoneInformation /t REG_DWORD /v RealTimeIsUniv
```

To verify this setting, type

```
reg query HKLM\System\CurrentControlSet\Control\TimeZoneInformation /s
```

2012-02-10: Microsoft Support released a bug warning and advises for the time being not to set *RealTimeIsUniversal*: System may become unresponsive during Daylight Saving Time (DST) changeover if RealTimeIsUniversal is set.

2013-03-04: Microsoft Support released Hotfix 2800213 for Windows Server 2008 and Windows 7, fixing a bug in the way hardware abstraction layer (HAL) uses the CMOS clock time, which could have led to 100% CPU utilization after a DST changeover if *RealTimeIsUniversal*=1.

2014-02-11: Microsoft Support released Update KB2922223 for Windows Server 2008 and Windows 7, which fixes the problem that with *RealTimeIsUniversal*=1 it was no longer possible to change the system time through the control panel.

Looks like we are very slowly getting there ...

Related references

- Search for RealTimeIsUniversal for various discussions that this article has stimulated since 2001.
- CMOS RAM
- Windows Time Provider Interface
- How Microsoft Windows NT 4.0 Handles Internal Time
- Olson tz database (a comprehensive well-maintained public domain UTC-to-local-time conversion package used in numerous operating systems)
- International Standard Date and Time Notation

Various Microsoft KnowledgeBase articles provide examples of the rather serious conceptual problems that the use of local time in Microsoft APIs has created:

- Time stamp changes with Daylight Savings
- Obtaining UTC from NTFS files
- SetLocalTime/GetLocalTime not the same if adjusting for Daylight Savings Time

The use of local time in the RTC is only the most easy to fix problem in a whole range of related issues. None of the above problems would have appeared had Windows followed the POSIX practice of using UTC everywhere and providing conversion to the local time in any specifyable time zone only as a completely separate library functionality. Local time should be completely irrelevant for the operation of an operating system, it is just a convenient display format for application user interfaces. Microsoft API architects might in that context also be interested in the Proposal for a New Time API for ISO C 200X that solves a range of other problems in the current POSIX and ISO C time API.

Thanks to the folks at Microsoft (Fabien Petitcolas, Rob Earhart) and on comp.protocols.time.ntp and tz who provided feedback on this memo.

Please don't hesitate to contact me if you have any related information, ideas, questions or suggestions. I'll keep this text up-to-date as new information arrives.

Markus Kuhn

created 2001-07-02 – last modified 2012-03-12 – http://www.cl.cam.ac.uk/~mgk25/mswish/ut-rtc.html