The end of time

(32bit edition)

Presented by

Arnd Bergmann

Date

March 25, 2015

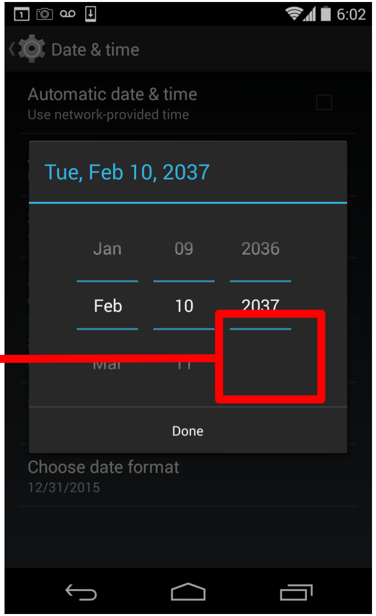
Event

Embedded Linux Conference

VALLEY

Date and Time settings on an Android phone (Nexus 4) today.

No year beyond 2037?



6:02

Date & time

Automatic date & time
Use network-provided time

Tue, Feb 10, 2037

Jan 09 2036

Feb 10 2037

Mar 11

Done

Choose date format
12/31/2015

Overview: The end of time

- Understanding the problem
- Finding a solution
- Fixing driver internals
- Fixing system calls
- Fixing user space
- Fixing ioctl
- Fixing file systems

Understanding the problem

2038 issue

- Unix/POSIX’s Y2K
- time_t representing number of seconds since Jan 1 1970.
- 32bit systems can represent dates from:
  - Dec 13 1901
  - Jan 19th 2038
- Less than 23 years to go

What does this mean for Linux?

- On Jan 19th 2038, time_t values overflow and go negative.
- Negative times for timers are considered invalid, so timers set beyond Jan 19 2038 fail
- Internal kernel timers set beyond Jan 19 2038 will never fire
- Until recently the kernel would hang as soon as time_t rolls negative

But we have 23 years!

That’s tons of time!
Folks will just upgrade to 64bits by then!

Problem with that...

Lots and lots of 32bit ARM devices being deployed today that may have 23+ year life spans

1992 Honda Civic

1992 wasn’t so long ago

- So maybe not a “classic” car, but these are still on the road.
- People expect their radio to still work
- Especially if they paid for the fancy in-dash infotainment system.

1992 Lamborghini Diablo

* Assuming you can still get battery replacements then….

Other long deployment life systems

- Security systems
- Utility monitoring sensors
- Satellites
- Medical devices
- Industrial fabrication machines

As embedded processors gain power, these are more likely to be running general-purpose kernels like Linux

Finding a solution

Crux of the issue

- Moving to 64 bit hardware isn’t a realistic answer for everyone
    - Even with 64 bit hardware, some users rely on 32 bit user space
- However, today’s 32 bit applications are terminally broken

OpenBSD precedent

- Converted time_t to long long in 2013:
  http://www.openbsd.org/papers/eurobsdcon_2013_time_t/
- Broke ABI, but “distro” is self-contained so limited compatibility damage
  - Lots of interesting thoughts there on the risks of compatibility support delaying conversion

NetBSD precedent

- Added a new system call API in 2008
- Kept ABI compatibility for existing binaries
- No option to build against old ABI
- Some user space (e.g. postgresql) broke after recompiling

Our strategy for Linux

- Build time:
  - support both 32-bit and 64-bit time_t
  - Leave decision up to libc
- Run time:
  - Like NetBSD, add a new 64-bit time_t ABI
  - Support both system call ABIs by default
  - Allow 32-bit time_t interface to be disabled

Kernel implications

- Have to change one bug at a time
- Hundreds of drivers
- 30-40 system calls
- Dozens of ioctl commands
- Incompatible changes for on-wire and on-disk data

Just to be clear

- This won’t solve *all* 2038 issues
- Just want to focus on solving the kernel issues and give a path for applications to migrate to.
- Applications likely do dumb things (which seemed reasonable at the time) w/ time_t
- If you’re ~40 years old, fixing this won’t hurt your supplemental retirement planning
    - There will still be lucrative contracts to validate and fix applications.

Current status

- Core timekeeping code already fixed
- OPW internship ongoing, lots of simple driver fixes, but more broken code gets added
- Linaro and others spending developer time
- Known broken files down from 783 to 725 since v3.15, lots more work to do

```
git grep -wl '\(time_t\|struct timespec\|struct timeval\)'
```

Fixing drivers internals

Fixing drivers, typical code

```
struct timeval start_time, stop_time;
do_gettimeofday(&start_time);
...
do_gettimeofday(&stop_time);
t = stop_time.tv_sec - start_time.tv_sec;
t *= 1000000;
if (stop_time.tv_usec < start_time.tv_usec)
        t -= start_time.tv_usec - stop_time.tv_usec;
else
        t += stop_time.tv_usec - start_time.tv_usec;
```

Example from
sound/pci/es1968.c

Fixing drivers, typical code

```
struct timeval start_time, stop_time;
do_gettimeofday(&start_time);
...
do_gettimeofday(&stop_time);
t = stop_time.tv_sec - start_time.tv_sec;
t *= 1000000;
if (stop_time.tv_usec < start_time.tv_usec)
       t -= start_time.tv_usec - stop_time.tv_usec;
else
       t += stop_time.tv_usec - start_time.tv_usec;
```

Trying to remove these

Fixing drivers, trivial fix

direct replacement type, using nanosecond resolution.

```
struct timespec64 start_time, stop_time;
do_getnstimeofday64(&start_time);
...
do_getnstimeofday64(&stop_time);
t = stop_time.tv_sec - start_time.tv_sec;
t *= 1000000000;
if (stop_time.tv_nsec < start_time.tv_nsec)
        t -= start_time.tv_nsec - stop_time.tv_nsec;
else
        t += stop_time.tv_nsec - start_time.tv_nsec;
t /= 1000;
```

Code was actually safe already but not obviously so.

Fixing drivers, trivial fix

```
struct timespec64 start_time, stop_time;
do_getnstimeofday64(&start_time);
...
do_getnstimeofday64(&stop_time);
t = stop_time.tv_sec - start_time.tv_sec;
t *= 1000000000;
if (stop_time.tv_nsec < start_time.tv_nsec)
        t -= start_time.tv_nsec - stop_time.tv_nsec;
else
        t += stop_time.tv_nsec - start_time.tv_nsec;
t /= 1000;
```

direct replacement type, using nanosecond resolution.

Code was actually safe already but not obviously so.

Possible overflow?

Fixing drivers, better fix

```
ktime_t start_time;
start_time = ktime_get();
...
t = ktime_us_delta(ktime_get(), start_time);
```

Using monotonic time also fixes concurrent settimeofday() calls

Efficient, safe and easy to use helper functions improve drivers further

Fixing system calls

System calls, example time()

```
SYSCALL_DEFINE1(time, time_t __user *, tloc)
{
        time_t i = get_seconds();
        if (put_user(i, tloc))
                return -EFAULT;
        return i;
}

#define __NR_time 13
```

System calls, example time()

```
SYSCALL_DEFINE1(time, time_t __user *, tloc)
{
        time_t i = get_seconds();
        if (put_user(i, tloc))
                return -EFAULT;
        return i;
}

#define __NR_time 13
```

Need to fix for 32-bit

System calls, example time()

```
SYSCALL_DEFINE1(time, __kernel_time64_t __user *, tloc)
{
        __kernel_time64_t i = get_seconds64();
        if (put_user(i, tloc))
                return -EFAULT;
        return i;
}

#define __NR_time       13
#define __NR_time64 367
```

Better, but now breaks compatibility

System calls, example time()

```
#ifdef CONFIG_COMPAT_TIME
COMPAT_SYSCALL_DEFINE1(time, compat_time_t __user *, tloc)
{
        compat_time_t i = (compat_time_t)get_seconds64();
        if (put_user(i, tloc))
                return -EFAULT;
        return i;
}
#endif
```

System calls, traditional types

```
typedef long __kernel_time_t;    /* user visible */
typedef __kernel_time_t time_t;  /* kernel internal */
```

System calls, intermediate types

```
typedef long __kernel_time_t;    /* user visible */
typedef __kernel_time_t time_t;  /* kernel internal */

#ifdef CONFIG_COMPAT_TIME
typedef s64 __kernel_time64_t;       /* user visible */
typedef s32 compat_time_t;       /* kernel internal */
#else
typedef long __kernel_time64_t;  /* internal HACK! */
#endif
```

System calls, final types

```
typedef long __kernel_time_t;    /* user visible */
typedef __kernel_time_t time_t;  /* kernel internal */
typedef s64 __kernel_time64_t;       /* user visible */

#ifdef CONFIG_COMPAT_TIME
typedef s32 compat_time_t;       /* kernel internal */
#endif
```

Fixing user space

Embedded distros

- Change libc to use 64-bit time_t
- Recompile everything
- ...
- Profit

Embedded distros

- Change libc to use 64-bit time_t
- Recompile everything
- ...
- Profit

- Caveat: ioctl

Embedded distros

- Change libc to use 64-bit time_t
- Recompile everything
- ...
- Profit

- Caveat: ioctl
- Caveat 2: programs hardcoding 32-bit types

Standard distros

- Need to provide backwards compatibility
- glibc to use symbol versioning
- multi-year effort

Standard distros

- Need to provide backwards compatibility
- glibc to use symbol versioning
- multi-year effort

- Any 32-bit standard distros remaining in 2038? Maybe Debian

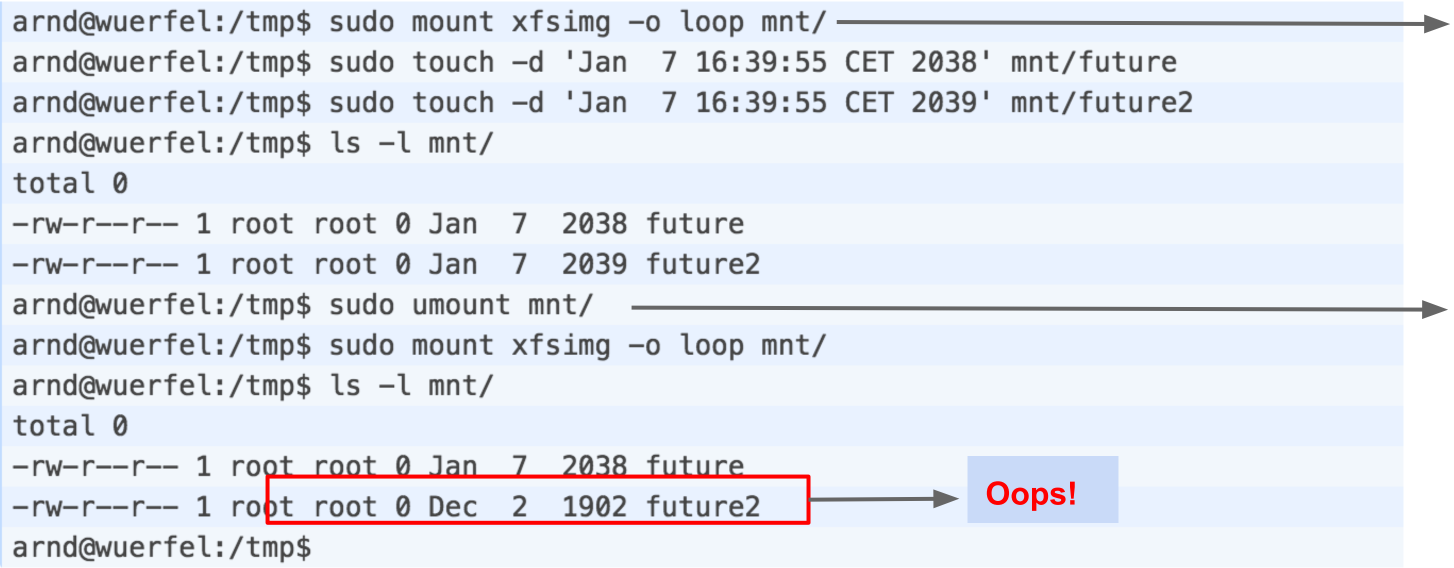
Fixing ioctl

Fixing ioctl commands in drivers

- Full audit of data structures needed
- Some user space needs source changes
- Recompiled user space tools may break on old kernels
- Some headers need #ifdef to know user time_t size

Fixing file systems

y2038 and filesystems



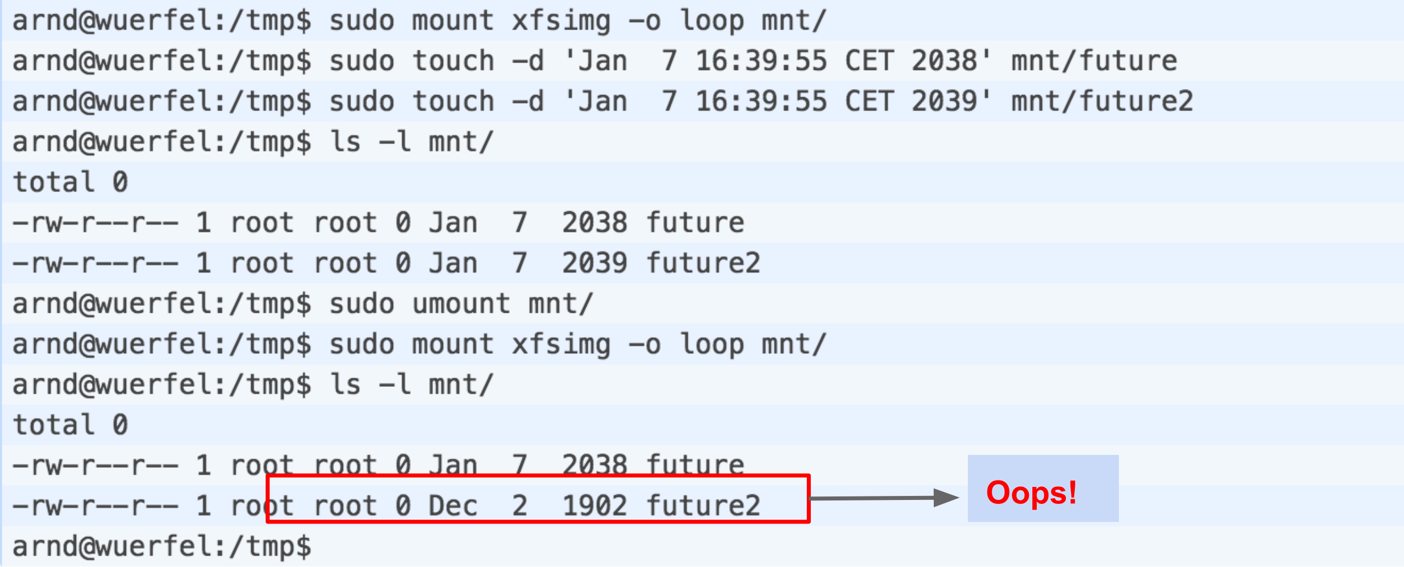
```
arnd@wuerfel:/tmp$ sudo mount xfsimg -o loop mnt/
arnd@wuerfel:/tmp$ sudo touch -d 'Jan  7 16:39:55 CET 2038' mnt/future
arnd@wuerfel:/tmp$ sudo touch -d 'Jan  7 16:39:55 CET 2039' mnt/future2
arnd@wuerfel:/tmp$ ls -l mnt/
total 0
-rw-r--r-- 1 root root 0 Jan  7  2038 future
-rw-r--r-- 1 root root 0 Jan  7  2039 future2
arnd@wuerfel:/tmp$ sudo umount mnt/
arnd@wuerfel:/tmp$ sudo mount xfsimg -o loop mnt/
arnd@wuerfel:/tmp$ ls -l mnt/
total 0
-rw-r--r-- 1 root root 0 Jan  7  2038 future
-rw-r--r-- 1 root root 0 Dec  2  1902 future2
arnd@wuerfel:/tmp$
```

Choosing xfs as an example.

Unmount followed by mount. Think of it as a reboot.

Oops!

y2038 and filesystems



```
arnd@wuerfel:/tmp$ sudo mount xfsimg -o loop mnt/
arnd@wuerfel:/tmp$ sudo touch -d 'Jan  7 16:39:55 CET 2038' mnt/future
arnd@wuerfel:/tmp$ sudo touch -d 'Jan  7 16:39:55 CET 2039' mnt/future2
arnd@wuerfel:/tmp$ ls -l mnt/
total 0
-rw-r--r-- 1 root root 0 Jan  7  2038 future
-rw-r--r-- 1 root root 0 Jan  7  2039 future2
arnd@wuerfel:/tmp$ sudo umount mnt/
arnd@wuerfel:/tmp$ sudo mount xfsimg -o loop mnt/
arnd@wuerfel:/tmp$ ls -l mnt/
total 0
-rw-r--r-- 1 root root 0 Jan  7  2038 future
-rw-r--r-- 1 root root 0 Dec  2  1902 future2
arnd@wuerfel:/tmp$
```

Oops!

This is a 64-bit system

Fixing filesystem timestamps

On-disk representation

- Up to the filesystem
- Example: Adding epochs in xfs. Reinterpret seconds field
- 8-bit padding → 255*136 years

Fixing filesystem timestamps

inode_operations

- change in-inode fields
- inode_time or timespec64?
- Rewriting getattr / setattr callbacks

Fixing filesystem timestamps

ioctl interface

- Also controlled by filesystem
- Rewrite ioctls to reinterpret time with epochs.
- update xfsprogs to understand new format

The end of time

The End

Special thanks:

John Stultz

Tina Ruchandani

Questions?

Backup slides

Discussed solutions

- Unsigned time_t
- New 64bit time_t ABI
- New kernel syscalls that provide 64bit time values

Unsigned time_t

- Have the kernel interpret time_t’s as unsigned values
- Would allow timers to function past 2038
    - Might work for applications that only deal with relative timers.
- Still problematic
    - Could modify glibc to properly convert time_t to “modern” string
    - Applications lose ability to describe pre-1970 events
- Could subtly change userspace headers?
    - Probably not a good idea

New Kernel ABI

- Define time_t as a long long
- Provide personality compat support for existing 32bit ABI
- Applications recompiled w/ new ABI would work past 2038, old ABI applications would work until 2038.

New Kernel ABI Drawbacks

- Still issues with application bad behavior
  - Internally casting time_t to longs
  - Storing time_t in 32bit file/protocol formats
- Have to implement new compat functions
  - Not a trivial amount of work
- New ABI is a big break, might want to “fix” more than just time_t
  - Might get lots of “riders” on the change

Add new gettime64() interfaces

- New time64_t type
- Also need interfaces for setting the time, setting timers, querying timers, sleep interfaces, etc.
- 30-40 syscalls take time_t
  - ioctls are even worse
- Lots of structures embed time_t