ktime accessors

Device drivers can read the current time using ktime_get() and the many related functions declared in linux/timekeeping.h. As a rule of thumb, using an accessor with a shorter name is preferred over one with a longer name if both are equally fit for a particular use case.

Basic ktime_t based interfaces

The recommended simplest form returns an opaque ktime_t, with variants that return time for different clock references:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 16)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get( void )

        CLOCK_MONOTONIC

        Useful for reliable timestamps and measuring short time intervals
        accurately. Starts at system boot time but stops during suspend.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 23)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get_boottime( void )

        CLOCK_BOOTTIME

        Like ktime_get(), but does not stop when suspended. This can be
        used e.g. for key expiration times that need to be synchronized
        with other machines across a suspend operation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 31)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get_real( void )

        CLOCK_REALTIME

        Returns the time in relative to the UNIX epoch starting in 1970
        using the Coordinated Universal Time (UTC), same as gettimeofday()
        user space. This is used for all timestamps that need to
        persist across a reboot, like inode times, but should be avoided
        for internal uses, since it can jump backwards due to a leap
        second update, NTP adjustment settimeofday() operation from user
        space.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 43)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get_clocktai( void )

         CLOCK_TAI

        Like ktime_get_real(), but uses the International Atomic Time (TAI)
        reference instead of UTC to avoid jumping on leap second updates.
        This is rarely useful in the kernel.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 51)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get_raw( void )

        CLOCK_MONOTONIC_RAW

        Like ktime_get(), but runs at the same rate as the hardware
        clocksource without (NTP) adjustments for clock drift. This is
        also rarely needed in the kernel.
```

nanosecond, timespec64, and second output

For all of the above, there are variants that return the time in a different format depending on what is required by the user:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 65)

Unknown directive type "c:function".

```
.. c:function:: u64 ktime_get_ns( void )
                u64 ktime_get_boottime_ns( void )
                u64 ktime_get_real_ns( void )
                u64 ktime_get_clocktai_ns( void )
                u64 ktime_get_raw_ns( void )

        Same as the plain ktime_get functions, but returning a u64 number
        of nanoseconds in the respective time reference, which may be
        more convenient for some callers.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 75)

Unknown directive type "c:function".

```
.. c:function:: void ktime_get_ts64( struct timespec64 * )
                void ktime_get_boottime_ts64( struct timespec64 * )
                void ktime_get_real_ts64( struct timespec64 * )
                void ktime_get_clocktai_ts64( struct timespec64 * )
                void ktime_get_raw_ts64( struct timespec64 * )

        Same above, but returns the time in a 'struct timespec64', split
        into seconds and nanoseconds. This can avoid an extra division
        when printing the time, or when passing it into an external
        interface that expects a 'timespec' or 'timeval' structure.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 86)

Unknown directive type "c:function".

```
.. c:function:: time64_t ktime_get_seconds( void )
                time64_t ktime_get_boottime_seconds( void )
                time64_t ktime_get_real_seconds( void )
                time64_t ktime_get_clocktai_seconds( void )
                time64_t ktime_get_raw_seconds( void )

        Return a coarse-grained version of the time as a scalar
        time64_t. This avoids accessing the clock hardware and rounds
        down the seconds to the full seconds of the last timer tick
        using the respective reference.
```

Coarse and fast_ns access

Some additional variants exist for more specialized cases:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 102)

Unknown directive type "c:function".

```
.. c:function:: ktime_t ktime_get_coarse( void )
                ktime_t ktime_get_coarse_boottime( void )
                ktime_t ktime_get_coarse_real( void )
                ktime_t ktime_get_coarse_clocktai( void )
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 107)

Unknown directive type "c:function".

```
.. c:function:: u64 ktime_get_coarse_ns( void )
                u64 ktime_get_coarse_boottime_ns( void )
                u64 ktime_get_coarse_real_ns( void )
                u64 ktime_get_coarse_clocktai_ns( void )
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 112)

Unknown directive type "c:function".

```
.. c:function:: void ktime_get_coarse_ts64( struct timespec64 * )
                void ktime_get_coarse_boottime_ts64( struct timespec64 * )
                void ktime_get_coarse_real_ts64( struct timespec64 * )
                void ktime_get_coarse_clocktai_ts64( struct timespec64 * )

        These are quicker than the non-coarse versions, but less accurate,
        corresponding to CLOCK_MONOTONIC_COARSE and CLOCK_REALTIME_COARSE
        in user space, along with the equivalent boottime/tai/raw
        timebase not available in user space.

        The time returned here corresponds to the last timer tick, which
        may be as much as 10ms in the past (for CONFIG_HZ=100), same as
        reading the 'jiffies' variable.  These are only useful when called
        in a fast path and one still expects better than second accuracy,
        but can't easily use 'jiffies', e.g. for inode timestamps.
        Skipping the hardware clock access saves around 100 CPU cycles
        on most modern machines with a reliable cycle counter, but
        up to several microseconds on older hardware with an external
        clocksource.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 132)

Unknown directive type "c:function".

```
.. c:function:: u64 ktime_get_mono_fast_ns( void )
                u64 ktime_get_raw_fast_ns( void )
                u64 ktime_get_boot_fast_ns( void )
                u64 ktime_get_real_fast_ns( void )

        These variants are safe to call from any context, including from
        a non-maskable interrupt (NMI) during a timekeeper update, and
        while we are entering suspend with the clocksource powered down.
        This is useful in some tracing or debugging code as well as
        machine check reporting, but most drivers should never call them,
        since the time is allowed to jump under certain conditions.
```

Deprecated time interfaces

Older kernels used some other interfaces that are now being phased out but may appear in third-party drivers being ported here. In

particular, all interfaces returning a 'struct timeval' or 'struct timespec' have been replaced because the tv_sec member overflows in year 2038 on 32-bit architectures. These are the recommended replacements:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 153)

Unknown directive type "c:function".

```
.. c:function:: void ktime_get_ts( struct timespec * )

        Use ktime_get() or ktime_get_ts64() instead.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 157)

Unknown directive type "c:function".

```
.. c:function:: void do_gettimeofday( struct timeval * )
                void getnstimeofday( struct timespec * )
                void getnstimeofday64( struct timespec64 * )
                void ktime_get_real_ts( struct timespec * )

        ktime_get_real_ts64() is a direct replacement, but consider using
        monotonic time (ktime_get_ts64()) and/or a ktime_t based interface
        (ktime_get()/ktime_get_real()).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 166)

Unknown directive type "c:function".

```
.. c:function:: struct timespec current_kernel_time( void )
                struct timespec64 current_kernel_time64( void )
                struct timespec get_monotonic_coarse( void )
                struct timespec64 get_monotonic_coarse64( void )

        These are replaced by ktime_get_coarse_real_ts64() and
        ktime_get_coarse_ts64(). However, A lot of code that wants
        coarse-grained times can use the simple 'jiffies' instead, while
        some drivers may actually want the higher resolution accessors
        these days.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\(linux-master)(Documentation)(core-api)timekeeping.rst, line 177)

Unknown directive type "c:function".

```
.. c:function:: struct timespec getrawmonotonic( void )
                struct timespec64 getrawmonotonic64( void )
                struct timespec timekeeping_clocktai( void )
                struct timespec64 timekeeping_clocktai64( void )
                struct timespec get_monotonic_boottime( void )
                struct timespec64 get_monotonic_boottime64( void )

        These are replaced by ktime_get_raw()/ktime_get_raw_ts64(),
        ktime_get_clocktai()/ktime_get_clocktai_ts64() as well
        as ktime_get_boottime()/ktime_get_boottime_ts64().
        However, if the particular choice of clock source is not
        important for the user, consider converting to
        ktime_get()/ktime_get_ts64() instead for consistency.
```