## Agenda

Clock and Time Management

## Time in Linux

Types of timing activities

- Keeping the current date and time
- Maintaining timers

## Time Representation

- Wall time (or real time)
- Process time
- Monotonic time

Ways to measure

- Relative time
- Absolute time

## Timing Hardware

System clock and Timers

- Real-time clock (RTC)
- Timestamp counter (TSC)
- Programmable interrupt timer (PIT)
- CPU local timer (in local APIC)
- High-precision event timer (HPET)
- ACPI power management timer (ACPI PMT)

The generic API for clock sources management
include/linux/clocksource.h →

## Calculating Elapsed Time

Time (seconds) = (counter value)/(clock frequency)

s64 clocksource_cyc2ns(u64 cycles, u32 mult, u32 shift) →

To evaluate mult and shift factors

void clocks_calc_mult_shift(u32 *mult, u32 *shift,…) →

```
struct clocksource *cs = &curr_clocksource;
cycle_t start = cs->read(cs);
/* things to do */
cycle_t end = cs->read(cs);
cycle_t diff = end – start;
duration = clocksource_cyc2ns(diff, cs->mult, cs->shift);
```

## System Timer

#include <.../param.h>
kernel/time/timer.c

The timer interrupt rate is called HZ. Default value for x86_64 is 1000 (1ms)

May be redefined with the kernel configuration option CONFIG_HZ
Supported values are: 100, 250, 300, 1000

```
#include <linux/sched.h>
```

For safety read 64-bit value jiffies we can use following:

```
u64 get_jiffies_64(void);

time_after(a, b);
time_before(a, b);
time_after_eq(a, b);
time_before_eq(a, b);

time_in_range(a, b, c);                //   a in range [b, c]
time_in_range_open(a, b, c);          //   a in range [b, c)
```

## Absolute Time

#include <linux/time.h> →

```
struct timespec {
        __kernel_time_t         tv_sec;                 /* seconds */
        long                    tv_nsec;                /* nanoseconds */
};

struct timeval {
        __kernel_time_t                 tv_sec;         /* seconds */
        __kernel_suseconds_t    tv_usec;        /* microseconds */
};

struct timezone {
        int             tz_minuteswest;         /* minutes west of Greenwich */
        int             tz_dsttime;             /* type of dst correction */
};
```

## Time delay

### #include <linux/delay.h>

Delay routines, using a pre-computed "loops_per_jiffy" value.
Please note that ndelay(), udelay() and mdelay() may return early for several reasons:

1. computed loops_per_jiffy too low (due to the time taken to execute the timer interrupt.)
2. cache behaviour affecting the time it takes to execute the loop function.
3. CPU clock rate changes.

```
void ndelay( unsigned long int nanoseconds );
void udelay( unsigned long int microseconds );
void mdelay( unsigned long int milliseconds );
```

## Timers

```
#include <linux/timer.h>
struct timer_list {
                struct list_head entry;
                unsigned long expires;
                void (*function)( unsigned long );
                unsigned long data;
              ...
              };


void init_timer( struct timer_list *timer );
struct timer_list TIMER_INITIALIZER( _function, _expires, _data );
void add_timer( struct timer_list *timer );
void mod_timer( struct timer_list *timer, unsigned long expires );
int del_timer( struct timer_list *timer );
```

## #include <linux/sched.h>

```
set_current_state( TASK_INTERRUPTIBLE );
schedule_timeout( delay );
```

## #include <linux/delay.h>

```
void msleep( unsigned int milliseconds );
unsigned long msleep_interruptible( unsigned int milliseconds );
void ssleep( unsigned int seconds );
```

## High resolution Timers

```
#include <linux/ktime.h>
#include <linux/hrtimer.h>


struct hrtimer {
              ...
                ktime_t   _expires;
                enum hrtimer_restart (*function)(struct hrtimer *);
              ...
              }


enum hrtimer_restart {
                HRTIMER_NORESTART,
                HRTIMER_RESTART,
              };

void hrtimer_init( struct hrtimer *timer, clockid_t which_clock, enum hrtimer_mode mode );
int hrtimer_start( struct hrtimer *timer, ktime_t tim, const enum hrtimer_mode mode );
extern int hrtimer_cancel( struct hrtimer *timer );
```

## Real Time Counter

```
#include <linux/rtc.h>
#include <uapi/linux/rtc.h>


struct rtc_time {
        int tm_sec;
        int tm_min;
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
        int tm_wday;
        int tm_yday;
        int tm_isdst;
};
```

## Home Reading

- O. Tsiliurik Linux Kernel Development Book: Ch 6
- Look at sources:
  - uapi/linux/rtc.h
  - linux/rtc.h