

Time



ECE 373

Prelims

- Questions on assignments, class?
 - Datasheets and marketing info, PCI
- Midterm?

Time

- Measuring
- Storing
- Waiting
- Working

Jiffies

- Time values

- `jiffies` – global kernel time tick
- `HZ` – number of jiffies per second
- (versus tickless kernel)

- Operations

- `msecs_to_jiffies(m)`, `jiffies_to_msecs(j)`
- `nsecs_to_jiffies(n)`, `jiffies_to_nsecs(j)`
- `time_after(a,b)`, `time_before(a,b)`

Low-level time

- `get_cycles()`
 - Arch independent CPU cycle counter
 - `#define'd` to 0 if not supported
 - Arch specifics underneath
 - Sparc 64: https://elixir.bootlin.com/linux/latest/source/arch/sparc/include/asm/timex_64.h#L16
 - X86: <https://elixir.bootlin.com/linux/latest/source/arch/x86/include/asm/tsc.h#L23>
- `rdtsc` – Time Stamp Counter register
 - CPU register in x86 and x86_64 since pentium
 - 64-bit counter of clock cycles
- Good for low level timing – e.g. code profiling

get_cycles() example

- Time an operation

```
u32 c_start, c_done, c_duration;  
u32 answer;
```

```
c_start = get_cycles();  
answer = do_some_timewasting_thing();  
c_done = get_cycles();
```

```
c_duration = c_done - c_start;  
msecs = (1000 * c_duration) / cycles_per_sec;
```

Time of Day

- `current_kernel_time()`
 - Updated by tick, running time-of-date, init'ed from RTC
- `do_gettimeofday()`
 - Read from HW, adjusted with nanosecs & arch specific tweaks
- `mktime()`
 - Takes min/sec/etc to make seconds since "epoch"
- RTC – i2c devices to keep Time of Day info
 - Standard i2c device interface
 - Standard RTC device interface

Time example

- Time since "the epoch" – Jan 1, 1970, 00:00am

```
struct timeval tv;  
struct rtc_time rt;  
unsigned long secs1, secs2;
```

```
do_gettimeofday(&tv);  
secs1 = tv->tv_sec;
```

```
rtc->get_time(rtc, &rt);  
secs2 = mktime(rt->tm_year, rt->tm_mon, ...);
```

```
if (secs1 != secs2)  
    printk(KERN_INFO "Different times: %d != %d\n",  
           secs1, secs2);
```


Delay

- Delayers
- Sleepers
- Timers
- Schedulers

Delayers

- `mdelay()`, `udelay()`, `ndelay()`
 - While loop spin on a counter, no scheduler action
 - <https://elixir.bootlin.com/linux/latest/source/include/linux/delay.h>
 - <https://elixir.bootlin.com/linux/latest/source/arch/x86/lib/delay.c>
 - Only way to get short period delays
 - Not very friendly for long periods
 - Could block jiffies update if interrupts disabled
- Become very useful later when locking, interrupting...

Delayers, nicely

- Scheduler/CPU friendly
 - `cpu_relax()` - arch specific, might not do anything
 - `schedule()` - give up timeslice
- `while (!is_device_finished())`
 - `schedule();`

Sleepers

- `msleep()`
 - Give up timeslices for specific number of millisecs
- `msleep_interruptible()`
 - Same, but stop if a signal is pending
- <https://elixir.bootlin.com/linux/latest/source/kernel/time/timer.c>

Timers

- **Basics**

- `timer_setup(t_var, t_callback, t_flags)`
- `mod_timer(t_var, interval)`
- `del_timer_sync(t_var)`

- **Callback function**

- **Called when timer expires**
- `timer_cb(struct timer_list *t)`

Deferred work

- Tasklets
 - Often 2nd half of interrupt handler
 - Run on same CPU as interrupt that scheduled it
 - Can be interrupted
 - Can run as high priority (not interrupted by much)
 - Meant for quick handling – no sleeping
- Workqueue
 - Normal process
 - Longer running processing – can sleep
 - Launch delays on deferred work

Tasklets

```
int irq;  
DECLARE_TASKLET(my_task, finish_interrupt, (ulong)&irq);  
  
void finish_interrupt(ulong data) {  
    printk(KERN_INFO "irq number = %d\n", *(int *)data);  
}  
  
void irq_handler() {  
    irq = interrupt_information;  
    tasklet_schedule(&my_task);  
}
```

Workqueue example

```
void worker_func(void *data) {  
    /* lots of processing */  
}
```

```
DECLARE_WORK(my_worker, worker_func, 42);
```

```
/* or do it yourself */
```

```
my_w_queue = create_workqueue("MyWork");
```

```
queue_delayed_work(my_w_queue, my_worker, 1*HZ);
```

```
/* share simplified queue access */
```

```
schedule_delayed_work(&my_worker, 1*HZ);
```


Reading

- LDD3 – Chapter 7
- ELDD – Chapter 2, pgs 31-38
- Linux src
 - ../Documentation/rtc.c

