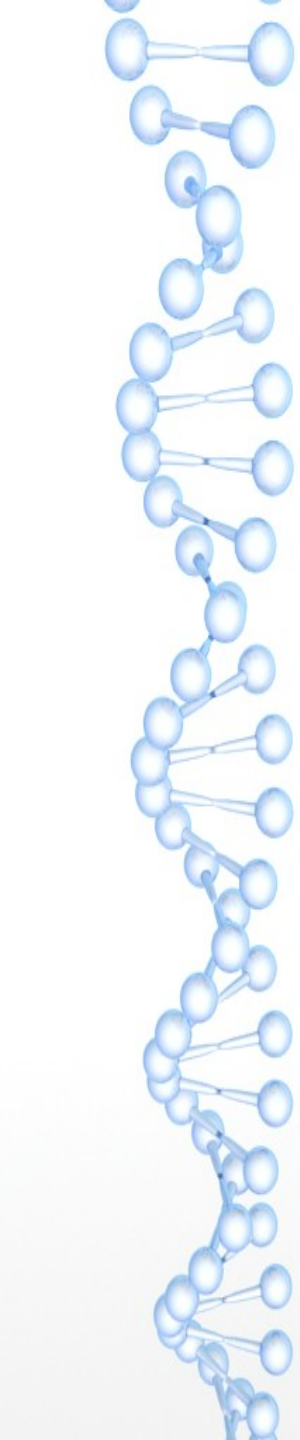
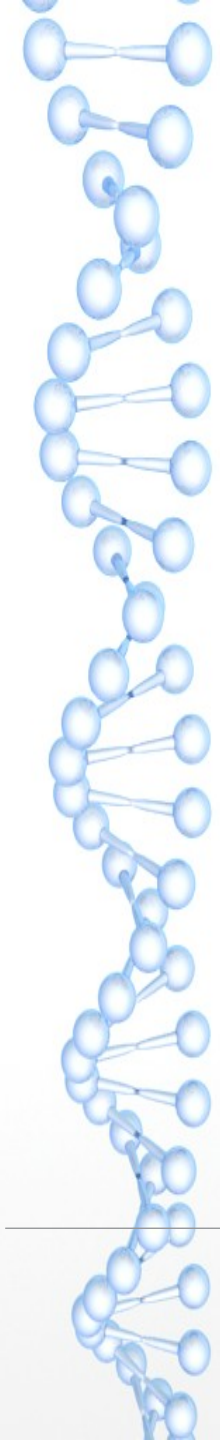


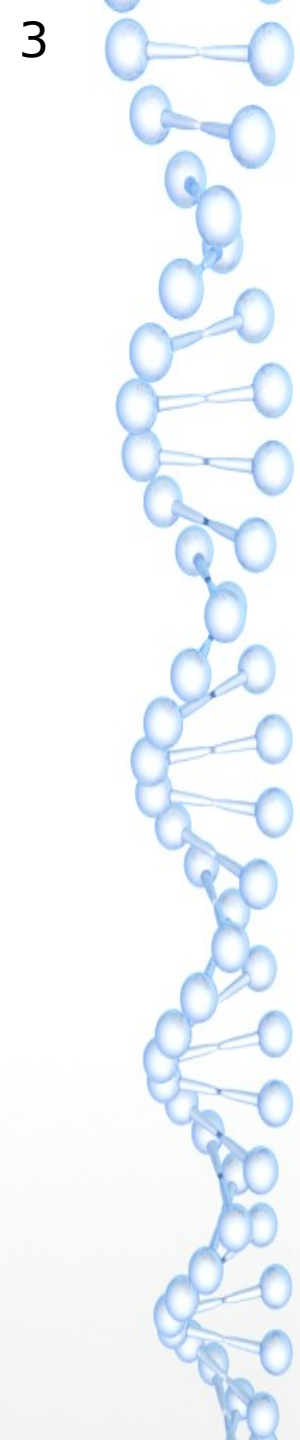
Android Power Management

Jacopo Mondi
March 2014



Summary

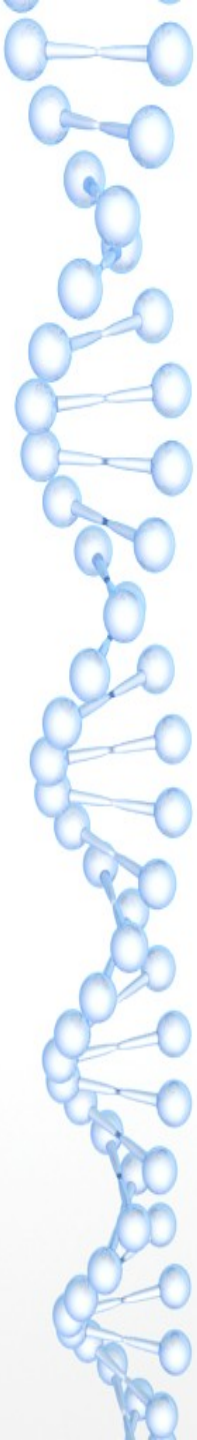
- Power Management Overview
- Frequency Scaling in Android/Linux
- Linux Power Management
- Android Specificities



Power Management Overview

- **Run-Time power consumption reduction**
  - Linux & Android:
  - Dynamic Frequency Scaling with CPUFreq

- **Power Management**
  - Linux & Android
  - Kernel Power Management
  - Linux:
  - CPUIdle
  - Runtime-PM
  - Android
  - Wakelocks
  - Earlysuspend/Lateresume



CPUFreq

Efficiently reducing system performances

CPUFreq

Standard Linux framework adopted by Android

- Separates scaling logic (governor) from implementation
- Governors implement scaling logic
  - Platform specific code performs actual scaling
- OPP = (Frequency, Voltage)
  - Governor sets frequency, platform code sets OPP
- Extensive debug interface through sysfs

CPUFreq

Governors can be switched at run-time

- Different governor are usually available
  - Performances
  - Conservative
  - On-Demand
  - ….
- Governors have parameters that modify their behavior
  - Thresholds
  - Available frequencies
  - Timing

CPUFreq

Android uses “Interactive”

- Designed to up-scale faster then on-demand to
- increase system responsiveness to user-interactions
  - Instead of sampling CPU each x msecs, ramps up after idle
  - Designed for latency critical environment, such as UI interactions

CPUFreq

Hands on:

```
# cd /sys/devices/system/cpu/cpu0/cpufreq
# cat scaling_governor
Interactive
# cat scaling_available_frequencies
396800 800000 1100000 1300000 1500000

# cat stats/trans_table
   From  :    To
         :    396800    800000   1100000   1300000   1500000
    396800:         0       323         0         0         2
    800000:       315         0        60         0         0
   1100000:         5        27         0        35         2
   1300000:         1         8         4         0        31
   1500000:         5        16         5         9         0

# cat stats/time_in_states
396800  145499
800000  7876
1100000 789
1300000 382
1500000 2285
```

CPUFreq

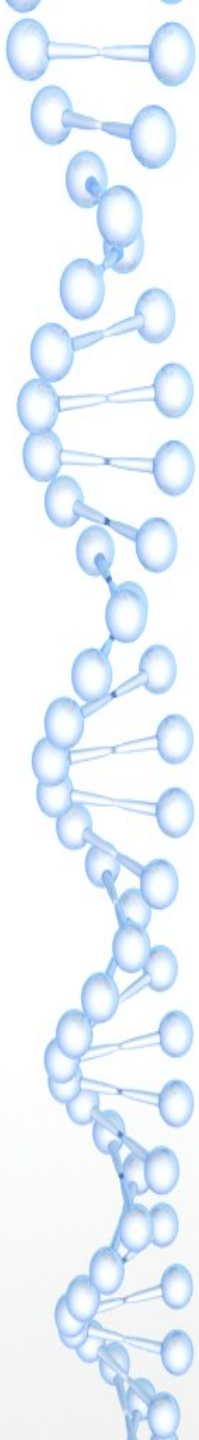
Hands on:

```
# cat scaling_cur_freq
 396800
```

Finger scroll the Android configuration settings page, make system 'active'....

```
# cat scaling_cur_freq
 1500000
```

Yes, it's quite 'interactive'



CPUFreq

Nice... Now show me the code (You know, talk is cheap blah blah..)

```
# ls -1 drivers/cpufreq/cpufreq*.c (governors)
drivers/cpufreq/cpufreq.c
drivers/cpufreq/cpufreq_conservative.c
drivers/cpufreq/cpufreq_interactive.c
drivers/cpufreq/cpufreq-nforce2.c
drivers/cpufreq/cpufreq_ondemand.c
drivers/cpufreq/cpufreq_performance.c
drivers/cpufreq/cpufreq_powersave.c
drivers/cpufreq/cpufreq_stats.c
drivers/cpufreq/cpufreq_userspace.c

# ls -1 drivers/cpufreq/*omap*.c (omap adaption layer)
Drivers/cpufreq/omap-cpufreq.c

# ls -1 arch/arm/mach-omap2/ (an author selection of)
arch/arm/mach-omap2/dvfs.c
arch/arm/mach-omap2/voltage.c
arch/arm/mach-omap2/abb.c
arch/arm/mach-omap2/frequency.c
arch/arm/mach-omap2/voltagedomains44xx_data.c
LOT OF NASTY STUFF HERE!!
```

CPUFreq

Nice... Now show me the code

Follow the *(pseudo)*code...

```
# cpufreq_interactive.c

static int cpufreq_interactive_speedchange_task(void *data)
{
      while(1){
            for_each_cpu(..){
                  ...
                  if (max_freq != pcpu->policy->cur)
                        __cpufreq_driver_target(pcpu->policy,
                                                max_freq,
                                                CPUFREQ_RELATION_H);
            }

      }
}
```

CPUFreq

Nice... Now show me the code

Follow the *(pseudo)*code...

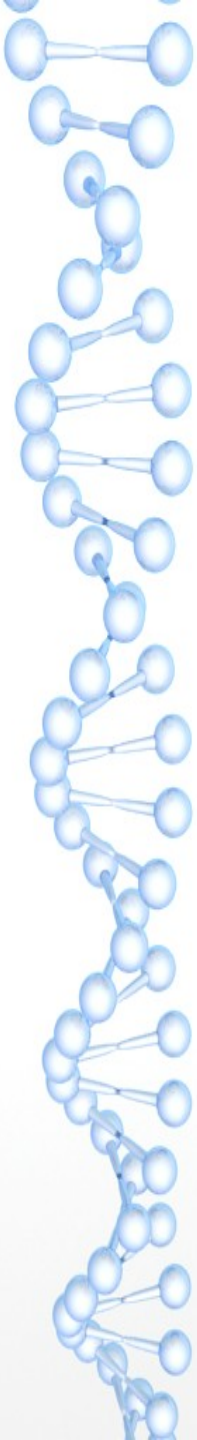
```
# cpufreq.c
int __cpufreq_driver_target(struct cpufreq_policy *policy,
                    unsigned int target_freq,
                    unsigned int relation)
{
      ...
      if (cpu_online(policy->cpu) && cpufreq_driver->target)
            retval = cpufreq_driver->target(policy, target_freq, relation);

      return retval;
}
EXPORT_SYMBOL_GPL(__cpufreq_driver_target);
```

CPUFreq

Nice... Now show me the code

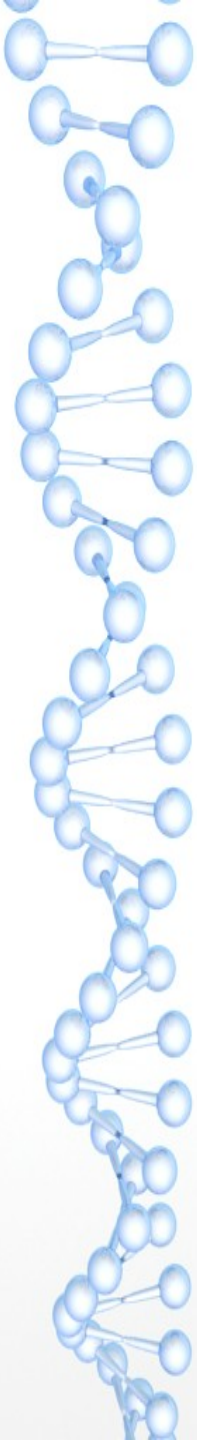
Follow the *(pseudo)*code...

```
# omap_cpufreq.c

static struct cpufreq_driver omap_driver = {
      .flags            = CPUFREQ_STICKY,
      .verify           = omap_verify_speed,
      .target           = omap_target,
      .get              = omap_getspeed,
      .init             = omap_cpu_init,
      .exit             = omap_cpu_exit,
      .name             = "omap",
      .attr             = omap_cpufreq_attr,
};

static int omap_cpufreq_scale(struct cpufreq_policy *policy,
                        unsigned int target_freq, unsigned int cur_freq,
                        unsigned int relation)
{
      mutex_lock(&omap_cpufreq_lock);
      current_target_freq = freq_table[i].frequency;
      /* This dives into platform code! */
      ret = omap_cpufreq_scale(policy, current_target_freq, policy->cur,
                        relation);
      mutex_unlock(&omap_cpufreq_lock);
```



Suspend

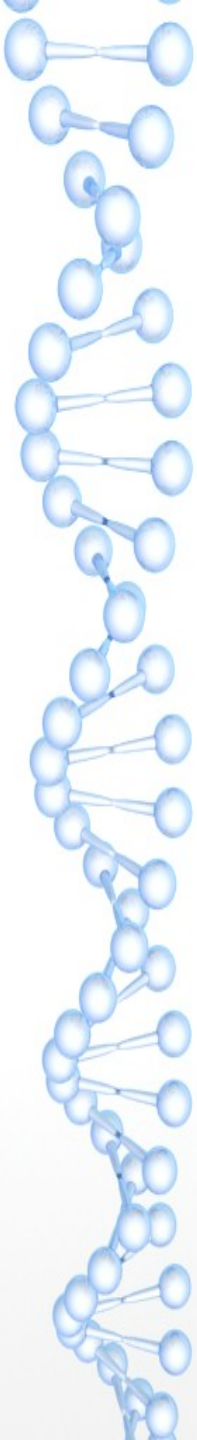
...sleeping never been so hard...



Diaspora

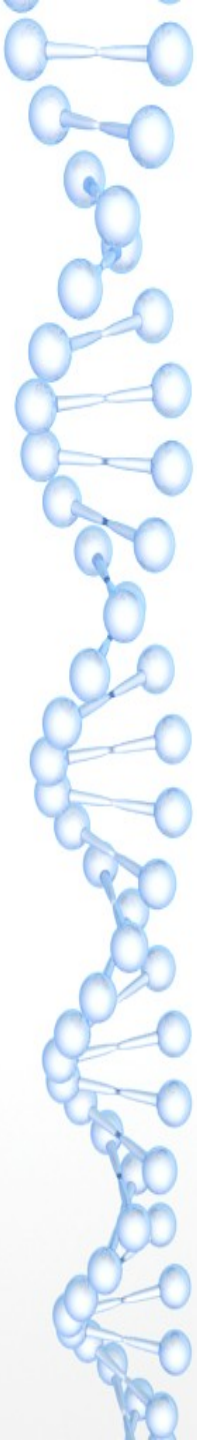
Android and standard Linux have diverged long time ago

- Different target devices
  - Linux has to be as generic as possible
  - Android has well defined use-cases
- Linux-only solutions were not ready at the time Android was designed
  - Android developers introduced their very own solutions
  - As well as Linux kernel community
- Two different approaches to sleep state management
  - Idle-based
  - Opportunistic



Diaspora

	Linux	Android
Frameworks	Runtime PM	Wakelocks
	CPUIdle	Earlysuspend/Lateresume
	Device driver PM	CPUIdle
		Device driver PM
Strategies	Suspend when Idle	Opportunistic suspend
	Idle devices can reduce their power consumption	Always tries to sleep
	User space triggers suspend execution	Kernel-based mechanism



Linux: Device Driver PM

Two power management models for device drivers

System Sleep:

Drivers enter low-power state during system-wide suspend operation.
These operations are usually device specific, and device drivers provide a set of standard callbacks to PM core, invoked during suspend operations

Runtime PM (Linux only)

Drivers provide an interface to reduce runtime power consumption.
In order to guarantee correctness of operations, the whole device tree is affected by runtime PM.
Linux-only mechanism, alternative to Android mechanisms.

Linux: Device Driver PM

Power management operations (include/linux/pm.h):

Drivers provides callbacks invoked by PM core during suspend and (eventually) runtime power management

```
struct dev_pm_ops {
      int (*prepare)(struct device *dev);
      void (*complete)(struct device *dev);
      int (*suspend)(struct device *dev);
      int (*resume)(struct device *dev);
      int (*freeze)(struct device *dev);
      int (*thaw)(struct device *dev);
      int (*poweroff)(struct device *dev);
      int (*restore)(struct device *dev);
      int (*suspend_late)(struct device *dev);
      int (*resume_early)(struct device *dev);
      int (*freeze_late)(struct device *dev);
      int (*thaw_early)(struct device *dev);
      int (*poweroff_late)(struct device *dev);
      int (*restore_early)(struct device *dev);
      int (*suspend_noirq)(struct device *dev);
      int (*resume_noirq)(struct device *dev);
      int (*freeze_noirq)(struct device *dev);
      int (*thaw_noirq)(struct device *dev);
      int (*poweroff_noirq)(struct device *dev);
      int (*restore_noirq)(struct device *dev);
      int (*runtime_suspend)(struct device *dev);
      int (*runtime_resume)(struct device *dev);
      int (*runtime_idle)(struct device *dev);
};
Run-Time only operations
```

Linux: Device Driver PM

Power management phases

- During system suspend/resume different phases are traversed by the system
- Each phase consists in the invocation of PM-related callbacks
- PM Callbacks have precedence compared to other frameworks' callbacks

Once phase commences when all the callbacks invoked in the preceding one ends

- The (exclusive) order of callback invocation is:
  - Device Driver
  - Device Type
  - Device Bus
  - Device Class

Linux: Device Driver PM

Suspend Phases

- Power management phases: *Prepare*
  - Prevent new devices from being registered
  - Device should not enter low power, but eventually prepare
- Power management phases: *Suspend*
  - *Quiesce device to prevent new I/O*
  - *Enter Low Power state*
- Power management phases: *Suspend_late*
  - *Eventually store device state after I/O has been disabled*
- Power management phases: *Suspend_noirq*
  - *Final callback, with interrupt disabled*

Not all callbacks are mandatory

- *If any of the above fails, system suspend is aborted*

Linux: Device Driver PM

Resume Phases

- Power management phases: *Resume_noirq*
  - Actions to be executed before interrupts for the device is enabled
- Power management phases: *Resume_early*
  - *Prepare for resume*
  - *Dual of suspend_late*
- Power management phases: *Resume*
  - *Restore normal activity, enable I/O*
- Power management phases: *Complete*
  - *Re-enable registration of child devices (for bus and classes)*

Hardware can be reset during suspend, driver should handle this.
Errors returned from these callbacks are ignored, system resume is never aborted

Linux: Device Driver PM

Power Domain Management

Power Domain

set of hw components sharing a resource, such as clock, power sources, voltage regulators.

Domains have policy manager associated:

Eg.
A clock manager within a domain takes into consideration joint constraints of the domain sub-components.

Components in a power domain have to be put in low power state together, as well as woke up at the same time

Linux: Device Driver PM

Power Domain Management

Domain Managers

- The `struct device` holds a pointer to a `dev_pm_domain` object, which provides a set of domain-wide power management callbacks (as it happens for drivers and classes or bus)

The power management domain callbacks, if defined for the given device, always take precedence over the callbacks provided by the device's subsystem (e.g. bus type)

- Relevant for SoCs and embedded platforms

Linux: Device Driver PM

CPU Idle states

Different idle states between active device and full suspend

- C1 = CPU0/1 ON; MPU ON; CORE ON
- C2 = CPU0/1 OFF; MPU INA; CORE INA
- C3 = CPU0/1 OFF; MPU CSWR; CORE CSWR
- C4 = CPU0/1 OFF; MPU OSWR; CORE OSWR
-
- Where:

OSWR: Open Switch Retention
Logic is lost for all the modules in the power domain except for the ones with Built in retention Flip Flops
CSWR: Closed Switch Retention
Logic is preserved for all modules

Linux: Device Driver PM

CPUIdle

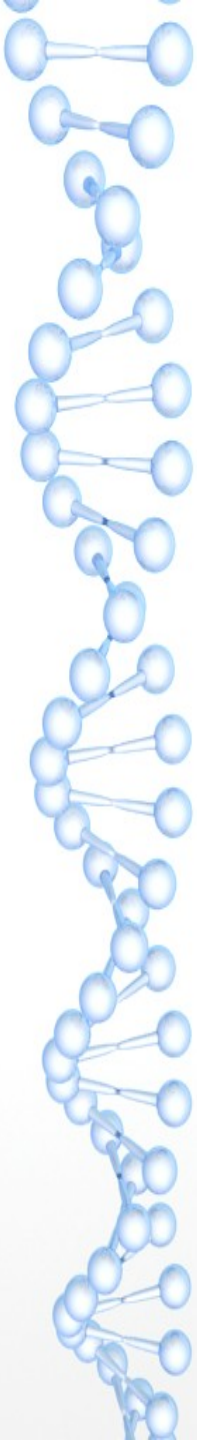
Framework to manage Idle states

Separates logic from implementation, as CPUFreq

Platform specific driver in
`arch/arm/mach-omap2/cpuidle-44xx.c`

Generic driver code in

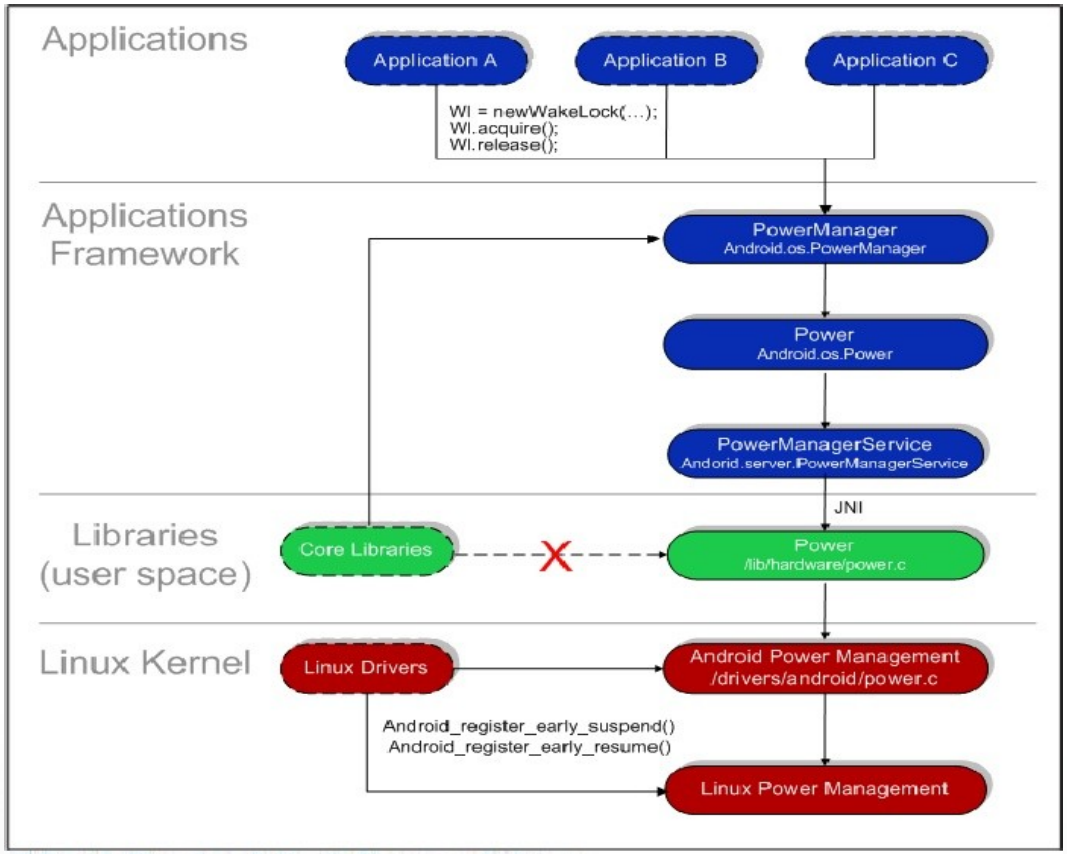
```
#ls drivers/cpuidle/*.c -1
drivers/cpuidle/coupled.c
drivers/cpuidle/cpuidle.c
drivers/cpuidle/driver.c
drivers/cpuidle/governor.c
drivers/cpuidle/sysfs.c
```



Android

Introducing locks, which once locked, lock the system from exiting wake state

Android: Power Management



Android: Power Management

PowerManager class

Provide power management features to applications and services

- Part of the Android framework, mediates between userspace and kernel interface, usually based on *sysfs*
- Application can require wakelocks from power manager:
- Very dangerous: badly written application can drain the battery
- A combination of flags determine which system components have to be kept “awake”

FLAGS	**CPU**	**SCREEN**	**KEYBOARD**
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Bright

Android: Power Management

The power HAL

HAL for power management

```
$find . -name power.c
./libhardware_legacy/power/power.c
./qcom/power/power.c
./libhardware/modules/power/power.c
```

Interact with drivers through sysfs

```
$cat ./libhardware_legacy/power/power.c
…
 const char * const NEW_PATHS[] = {
    "/sys/power/wake_lock",
    "/sys/power/wake_unlock",
};
…
```

Writes to these files in order to obtain or release a wakelock

Android: Power Management

Wakelocks for applications

Wakelocks prevent system from going in suspend mode

- Two locking methods:
- *Unlimited*: wakelock has to be released explicitly
- *Timed*: wakelock expires after a timeout

- Two wakelock types:
  - WAKE_LOCK_SUSPEND: prevents a full system suspend.
  - WAKE_LOCK_IDLE: low-power states, which often cause large interrupt latencies or that disable a set of interrupts, will not be entered from idle until the wakelocks are released.

Locking a wakelock during a suspend operation, aborts the suspend sequence if *suspend_late* has not been reach

This has unconventional consequences on wakeup sequences

Android: Power Management

Wakelocks on kernel side

- Device drivers can hold wakelocks as well
- In fact, they often have to do that, in order to pass wake up notifications to user space

- Kernel API:

```
struct wakelock wakelock;
wake_lock_init(&wakelock, WAKE_LOCK_SUSPEND, "wakelockname");

wake_lock(&wakelock);
wake_lock_timeout(&wakelock, HZ);

wake_unlock(&wakelock);

wake_lock_destroy(&wakelock);
```

Android: Power Management

Wakelocks on kernel side

`kernel/power/wakelock.c`

- One 'suspend' task declared (workqueue item), scheduled when
  - Last locked wakelock is released
  - Last locked wakelock expires (Timed wakelock)
  - Another condition when locking that is not clear at all to me :)

```
....
     if (del_timer(&expire_timer))
                    if (debug_mask & DEBUG_EXPIRE)
                         pr_info("wake_lock: %s, stop expire timer\n",
                              lock->name);
               if (expire_in == 0)
                    queue_work(suspend_work_queue, &suspend_work);
          }
     }
     ….
```

Android: Power Management

Wakelocks on kernel side

`kernel/power/wakelock.c`

- The suspend work

```
        …
    if (has_wake_lock(WAKE_LOCK_SUSPEND)) {
         if (debug_mask & DEBUG_SUSPEND)
              pr_info("suspend: abort suspend\n");
         return;
    }

    entry_event_num = current_event_num;
    sys_sync();
    if (debug_mask & DEBUG_SUSPEND)
         pr_info("suspend: enter suspend\n");
    getnstimeofday(&ts_entry);
    ret = pm_suspend(requested_suspend_state);
```

pm_suspend calls standard linux *enter_state(PM_SUSPEND_MEM)*
This will enter the suspend state, invoking the proper callbacks on each device/subsystem, as explained in previous slides

Android: Power Management

Earlysuspend

- Introduced alongside with Wakelocks
- Basically, a series of callbacks that drivers register to the PM core
- Drivers gets notification of userspace writing to
- `/sys/power/request_state`
- Modify the user-visible power state of the device
- EARLY_SUSPEND_LEVEL_BLANK_SCREEN
- EARLY_SUSPEND_LEVEL_STOP_DRAWING
- EARLY_SUSPEND_LEVEL_DISABLE_FB
- EARLY_SUSPEND_LEVEL_STOP_INPUT
- Applies to 'visible' devices (Display, sensors, framebuffer...)
- Dual to 'late resume' feature

Android: Power Management

Earlysuspend core

`kernel/power/earlysuspend.c`

- Main function:
  - `request_suspend_state(suspend_state_t new_state);`

- Schedule execution of earlysuspend/lateresume work items

```
if (!old_sleep && new_state != PM_SUSPEND_ON) {
     state |= SUSPEND_REQUESTED;
     queue_work(suspend_work_queue, &early_suspend_work);
} else if (old_sleep && new_state == PM_SUSPEND_ON) {
     state &= ~SUSPEND_REQUESTED;
     wake_lock(&main_wake_lock);
     queue_work(suspend_work_queue, &late_resume_work);
}
```

Android: Power Management

early_suspend_work

- Walk the list of callbacks registered by device drivers

```
mutex_lock(&early_suspend_lock);
spin_lock_irqsave(&state_lock, irqflags);
if (state == SUSPEND_REQUESTED)
     state |= SUSPENDED;
else
     abort = 1;
spin_unlock_irqrestore(&state_lock, irqflags);

…

list_for_each_entry(pos, &early_suspend_handlers, link) {
     if (pos->suspend != NULL) {
          if (debug_mask & DEBUG_VERBOSE)
               pr_info("early_suspend: calling %pf\n", pos->suspend);
          pos->suspend(pos);
     }
}
mutex_unlock(&early_suspend_lock);
```

Android: Power Management

late_resume_work

- Walk the list of callbacks registered by device drivers

```
mutex_lock(&early_suspend_lock);
spin_lock_irqsave(&state_lock, irqflags);
if (state == SUSPENDED)
     state &= ~SUSPENDED;
else
     abort = 1;
spin_unlock_irqrestore(&state_lock, irqflags);

…

list_for_each_entry_reverse(pos, &early_suspend_handlers, link) {
     if (pos->resume != NULL) {
          if (debug_mask & DEBUG_VERBOSE)
               pr_info("late_resume: calling %pf\n", pos->resume);
          pos->resume(pos);
     }
}
```

Android: Power Management

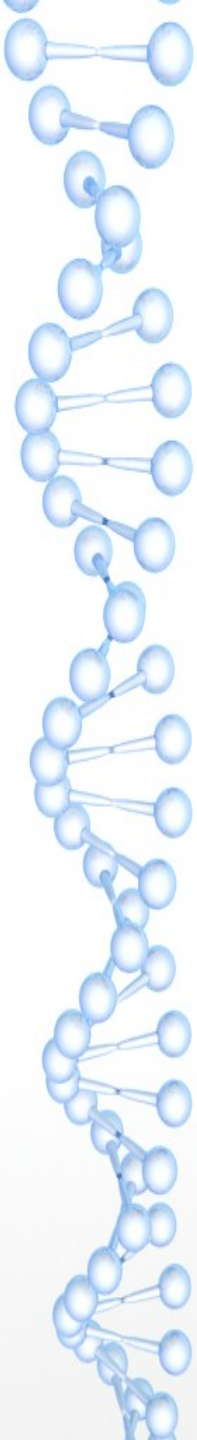
Driver: callback register

Driver API:

```
void register_early_suspend(struct early_suspend *handler)
void unregister_early_suspend(struct early_suspend *handler)
```

- Client Drivers: `drivers/leds/ledtrig_sleep.c`

```
static void ledtrig_sleep_early_suspend(struct early_suspend *h);
static void ledtrig_sleep_early_resume(struct early_suspend *h);

static struct early_suspend ledtrig_sleep_early_suspend_handler = {
        .suspend = ledtrig_sleep_early_suspend,
        .resume = ledtrig_sleep_early_resume,
};

static int __init ledtrig_sleep_init(void)
{
        led_trigger_register_simple("sleep", &ledtrig_sleep);
        register_pm_notifier(&ledtrig_sleep_pm_notifier);
        register_early_suspend(&ledtrig_sleep_early_suspend_handler);
        return 0;
}
```



References

Cool stuff in here

References

CPUFreq

http://bamboopuppy.com/changing-androids-cpu-frequency/

- http://bamboopuppy.com/android-cpu-frequency-using-interactive-governor/
- https://code.google.com/p/milestone-overclock/wiki/SmashingTheAndroidKernel
- https://www.kernel.org/doc/Documentation/cpu-freq/cpufreq-stats.txt

References

Power Management

http://elinux.org/OMAP_Power_Management
http://www.docstoc.com/docs/36734923/OMAP3430-Linux-Power-Management---PowerPoint
http://elinux.org/Android_Power_Management
http://developer.android.com/reference/android/os/PowerManager.html
http://2013.efyexpo.com/wp-content/uploads/2013/03/_2_Android_Power_Management_EFY.pdf
http://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki2.pdf
http://elinux.org/images/0/08/ELC-2010-Hilman-Runtime-PM.pdf
http://bamboopuppy.com/android-suspend-a-brief-overview/
http://forum.xda-developers.com/wiki/Wakelocks
http://www.cs.rochester.edu/~sandhya/csc256/seminars/peter_android.pdf
http://www.scribd.com/doc/22685058/android-power-management
http://www.kandroid.org/online-pdk/guide/power_management.html
http://www.kandroid.org/online-pdk/guide/early_suspend.html
https://patchwork.kernel.org/patch/58064/ (Motorola Quickwakeup)
https://community.freescale.com/thread/261901

- And of course, the Linux kernel source code, and AOSP sources