Android’s Wake Locks

Linux Device Drivers

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

The Problem

We need:

- Minimal power consumption
- Safe “opportunistic suspend”
- Different degrees of “suspend”

Demand-based power management

The Problem

Android's answer:

- "Wake locks"
- Android-specific API
- Not found in `kernel.org` kernels
- (Not likely to *ever* be found there)

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

Wake Locks

In a nutshell...

- `WAKE_LOCK_IDLE` and `WAKE_LOCK_SUSPEND`
- Just like platform suspend/resume, but meant to be interruptible
- “Early” suspend methods, in addition to normal ones
- If you understand platform suspend, you understand wake locks!

Platform Suspend and Resume

Platform suspend:

- Kernel invokes all driver `suspend()` methods
- If any `suspend()` returns an error, the suspend is aborted
- Only a “wakeup source” can resume a suspended platform

```
# echo "mem" > /sys/power/state
pm_suspend(PM_SUSPEND_MEM);
```

Wake Lock-Driven Suspend

If a new lock is taken during the wait:

- Kernel invokes all driver `late_resume()` methods
- The “suspend” is effectively aborted

Implements a new power management “state”

- Or, perhaps, a “fire and forget/abort” platform suspend

Wake Lock-Driven Suspend

“Isn’t that just an interruptible suspend?”

- Yes, and no

“Isn’t that just runtime-pm?”

- Yes, and no

Wake Lock-Driven Suspend

Not an interruptible suspend:

- Intended to be entered frequently
- Interrupted due to user activity, not driver error
- Likely to be interrupted, or to proceeed after a delay
- Has to carefully “race” with userspace

Wake Lock-Driven Suspend

Not “runtime power management”:

- Invokes more than just device-related events
- (Regulators vs. PMIC vs. CPU idle opcodes, for example)
- Aborts aren’t necessarily related to device activity

(Referring here to the so-named kernel API)

Wake Lock Driver API

What this means for driver authors:

- (Not a lot, actually)

Wake Lock Driver API

```
#if defined(CONFIG_HAS_EARLYSUSPEND)
#include <linux/earlysuspend.h>
#endif

struct tsc2007 {
  ...
#if defined(CONFIG_HAS_EARLYSUSPEND)
  struct early_suspend early;
#endif
  ...
};
```

Wake Lock Driver API

```
/* the usual suspend() handler */
int tsc2007_suspend(struct device *dev)
{
  /* the usual suspend code */
  ...
}
```

Wake Lock Driver API

```
#ifdef CONFIG_HAS_EARLYSUSPEND
void tsc2007_early_suspend(struct early_suspend *e)
{
  struct tsc2007 *ts
     = container_of(e, struct tsc2007, early);
  tsc2007_suspend(&ts->client->dev);
}
#endif
```

Wake Lock Driver API

Yikes!

- Where is error handling?
- Platform suspend method must be idempotent!

Wake Lock Driver API

```
#ifdef CONFIG_HAS_EARLYSUSPEND
void tsc2007_late_resume(struct early_suspend *e)
{
  struct tsc2007 *ts
     = container_of(e, struct tsc2007, early);
  tsc2007_resume(&ts->client->dev);
}
#endif
```

Wake Lock Driver API

```
int tsc2007_probe(...)
{
  ...
#ifdef CONFIG_HAS_EARLYSUSPEND
  ts->early.level = EARLY_SUSPEND_LEVEL_BLANK_SCREEN;
  ts->early.suspend = tsc2007_early_suspend;
  ts->early.resume = tsc2007_late_resume;
  register_early_suspend(&ts->early);
#endif
  ...
}
```

Controlling Order

```
#define TSC_SUSPEND_LEVEL 1

...
ts->early.level = EARLY_SUSPEND_LEVEL_BLANK_SCREEN
   + TSC_SUSPEND_LEVEL;
...
```

Android’s Wake Locks

Linux Device Drivers

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer