Power Management from Linux Kernel to Android

黃俊維
R97944026
資訊網路與多媒體所

王博榮
R97942139
電信工程研究所

1. Introduction

For normal desktop computer, power management (PM) is used to reduce power consumption and reduce cooling requirements. Lower power consumption means lower heat dissipation, which increases system stability, and less energy use, which saves money and reduces the impact on the environment. For mobile device and embedded system device, it's much more important because the battery power is very limited. Nowadays, android phone and iPhone are more and more pervasive. There are more and more sensors and I/O in mobile device that can be used to improve the effectiveness of PM. The PM needs to be tuned for new mobile device's need. In this survey, we want to not only know the power management system used before, but also want to compare them with the design of Android PM.

2. How does power management system work?

One power management standard for computers is ACPI, which supersedes APM. All recent (consumer) computers have ACPI support. Why ACPI has more advantage than APM? We'll write a brief introduction both of them and compare the difference.

APM (Advanced Power Management)

APM consists of one or more layers of software that support power management in computers with power manageable hardware. APM defines the hardware independent software interface between hardware-specific power management software and an operating system power management policy driver. It masks the details of the hardware, allowing higher-level software to use APM without any knowledge of the hardware interface.

The APM software interface specification defines a layered cooperative environment in which applications, operating systems, device drivers and the APM BIOS work together to reduce power consumption. In brief, APM can extend the life of system batteries and thereby increases productivity and system availability.

ACPI (Advanced Configuration & Power Interface)

The ACPI specification was developed to establish industry common interfaces enabling robust operating system (OS)-directed motherboard device configuration and power management of both devices and entire systems. Different from APM, ACPI allows control of power management from within the operating system. The previous industry standard for power management, APM, is controlled at the BIOS level. APM is activated when the system becomes idle. The longer the system idles, the less power it consumes (e.g. screen saver vs. sleep vs. suspend). In APM, the operating system has no knowledge of when the system will change power states.
There are several software components that ACPI has:

- A subsystem which controls hardware states and functions that may have previously been in the BIOS configuration
  These states include:
  - Thermal control
  - Motherboard configuration
  - Power states (sleep, suspend)
- a policy manager, which is software that sits on top of the operating system and allows user input on the system policies
- The ACPI also has device drivers those control/monitor devices such as a laptop battery, SMBus (communication/transmission path) and EC (embedded controller).

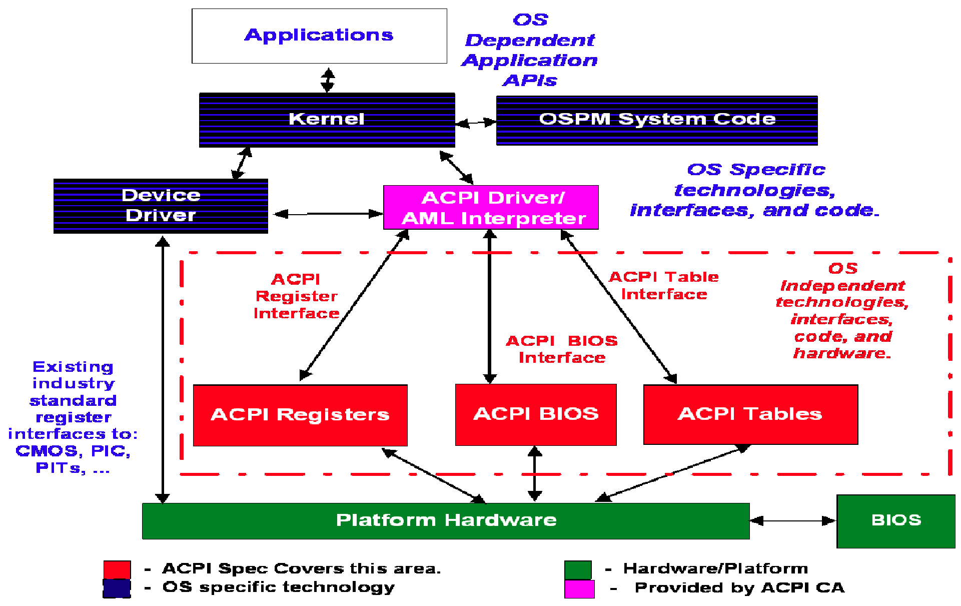

Applications
OS Dependent Application APIs
Kernel
OSPM System Code
Device Driver
ACPI Driver/ AML Interpreter
OS Specific technologies, interfaces, and code.
ACPI Register Interface
ACPI Table Interface
OS Independent technologies, interfaces, code, and hardware.
ACPI BIOS Interface
Existing industry standard register interfaces to: CMOS, PIC, PITs, ...
ACPI Registers
ACPI BIOS
ACPI Tables
Platform Hardware
BIOS
- ACPI Spec Covers this area.
- OS specific technology
- Hardware/Platform
- Provided by ACPI CA

Figure 2.1 CPI architecture

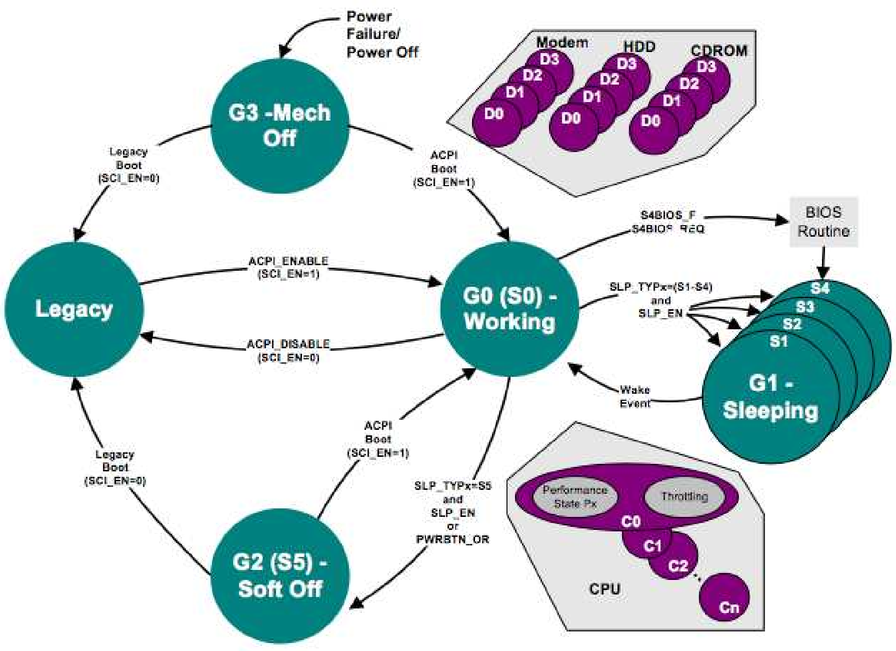
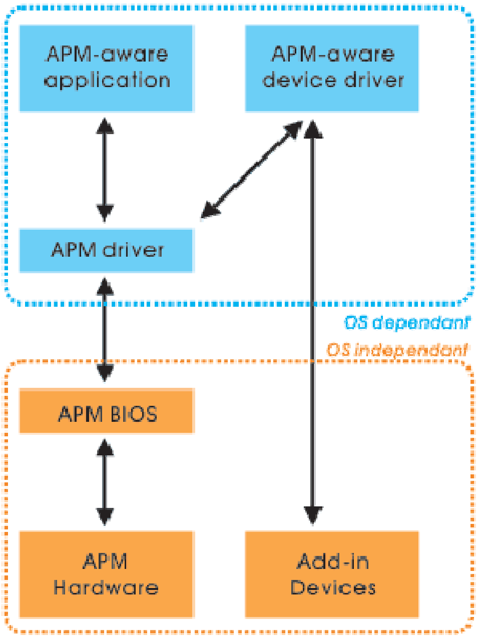
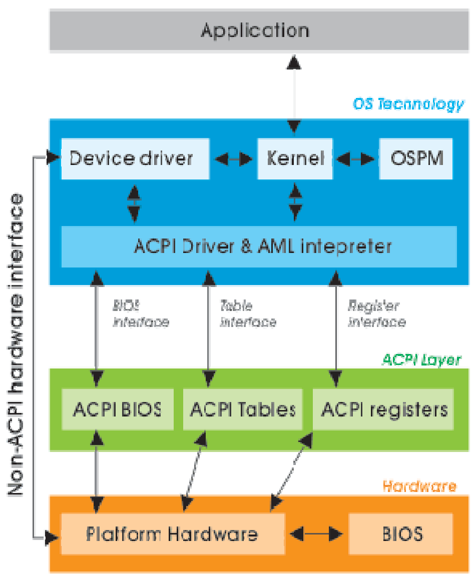
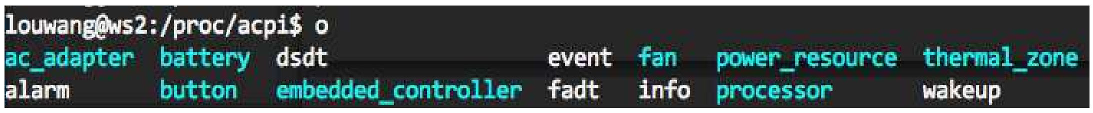


Figure 2.2 CPI power state transition diagram

Table 2.1 comparison between APM and ACPI

APM	ACPI
 APM-aware application APM-aware device driver APM driver OS dependant OS independant APM BIOS APM Hardware Add-in Devices	 Application OS Tecnnology Device driver Kernel OSPM ACPI Driver & AML intepreter BIOS interface Table interface Register interface ACPI Layer ACPI BIOS ACPI Tables ACPI registers Hardware Platform Hardware BIOS Non-ACPI hardware interface
o. Control resides in BIOS. o. Uses activity timeouts to determine when to power down a device o. Bios rarely used in embedded systems o. Makes power-management decisions without informing OS or individual applications o. No knowledge off add-in cards or new devices (e.g. USB, IEEEE 1394)	o. Control divided between BIOS and OS o. Decisions managed through the OS. Enables sophisticated power policy for general-purpose computers with standard usage patterns and hardware o. No knowledge of device-specific scenarios (e.g. Need to provide predictable response times or to respond to critical events over extended period) o. More power state means more specific adjustment to save power.



```
louwang@ws2:/proc/acpi$ o
ac_adapter  battery  dsdt                 event  fan   power_resource  thermal_zone
alarm       button   embedded_controller  fadt   info  processor       wakeup
```

Figure 2.3 ACPI Monitor Usages

We can either use “acpi -V” or look in each of the acpi files individually for information about our system. Check in the /proc/acpi directory for various things of importance. If you want to check your battery we can read the following file like this: **cat/proc/acpi/battery/BAT1/state**.

3. The Concept of Android power management

First of all, Android OS design is based on Linux kernel. Linux has its own power management that we have described in previous section. The following diagram (Figure 3.1) shows the main components of the Android OS.

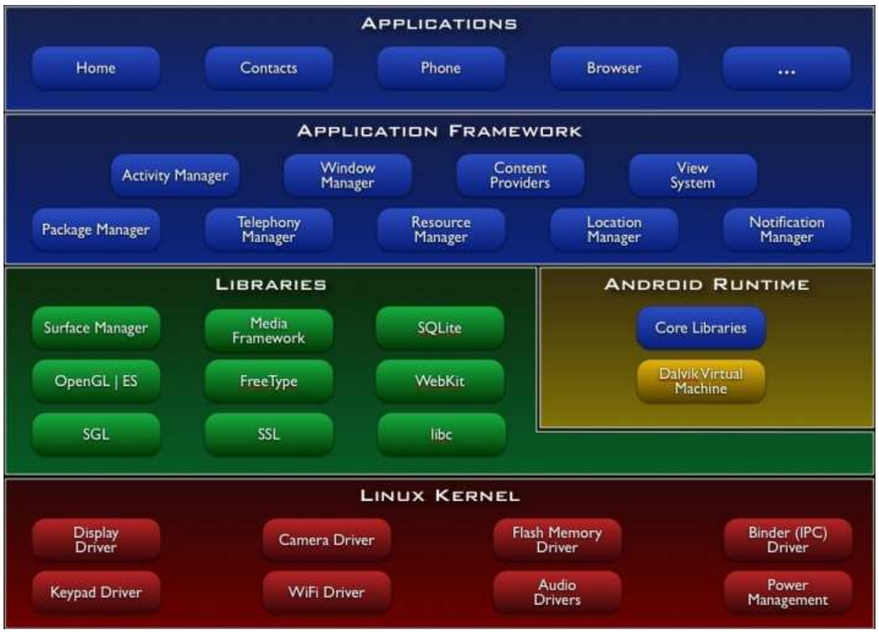
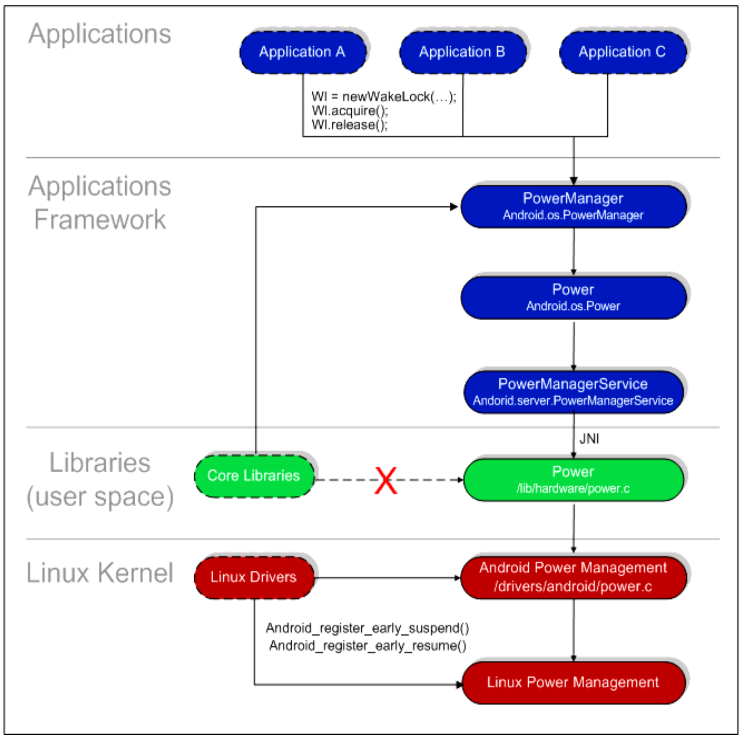


Figure 3.1 Android architecture

Android inherits many kernel components from Linux including power management component. Original power management of Linux is designed for personal computers, so there are some power saving status such as suspend and hibernation. However, these mechanisms of Linux PM do not satisfied and suitable for mobile devices or embedded systems. Mobile devices such as cell phones are not as same as PCs that have unlimited power supply. Because mobile devices have a hard constraint of limited battery power capacity, they need a special power management mechanism. Therefore, Android has an additional methodology for power saving.

The implementation of Android power management was sitting on top of Linux Power Management. Nevertheless, Android has a more aggressive Power Management policy than Linux, in which app and services must request CPU resource with "wake locks" through the Android application framework and native Linux libraries in order to keep power on, otherwise, Android will shut down the CPU.



```
Wl = newWakeLock(...);
Wl.acquire();
Wl.release();
```

Figure 3.2 Android Power Management.

Refer to Figure 3.2, Android try not to modify the Linux kernel and it implements an applications framework on top of the kernel called Android Power Management Applications Framework. The Android PM Framework is like a driver. It is written by Java which connects to Android power driver through JNI. However, what is JNI? JNI (Java Native Interface) is a framework that allows Java code running in a Java Virtual Machine (JVM) to call native C applications and libraries. Through JNI, the PM framework written by Java can call function from libraries written by C.

Android PM has a simple and aggressive mechanism called “Wake locks”. The PM supports several types of “Wake locks”. Applications and components need to get “Wake lock” to keep CPU on. If there is no active wake locks, CPU will turn off. Android supports different types of “Wake locks” (Table 3.1).

Table 3.1 Different wake locks of Android PM.

Wake Lock Type
ACQUIRE_CAUSES_WAKEUP
FULL_WAKE_LOCK
ON_AFTER_RELEASE
PARTIAL_WAKE_LOCK
SCREEN_BRIGHT_WAKE_LOCK
SCREEN_DIM_WAKE_LOCK

Currently Android only supports screen, keyboard, buttons backlight, and the brightness of screen. Because of full usage of CPU capability, it does not support suspend and standby mode. The following diagram shows how Android PM works. Through the framework, user space applications can use "PowerManger" class to control the power state of the device. We will introduce more details about how to implement them in applications later.

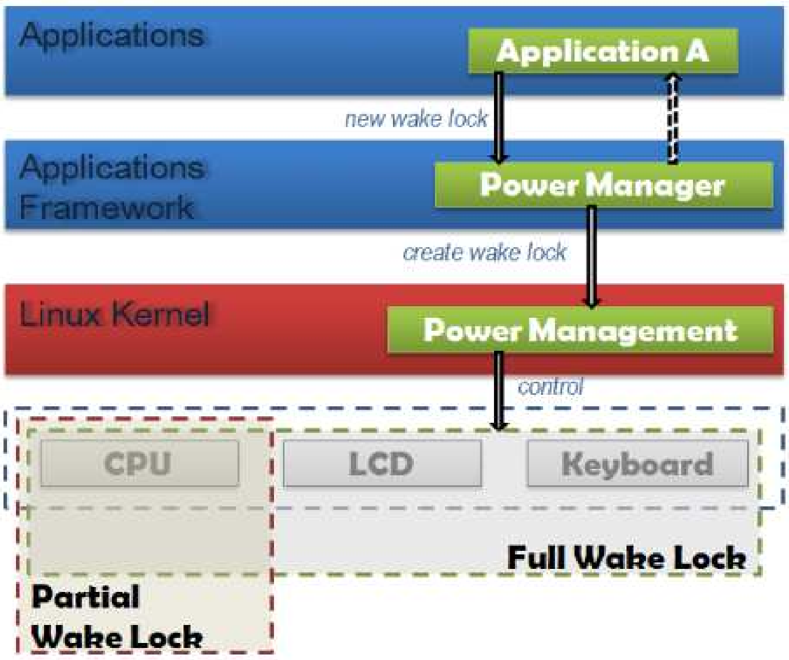


Figure 3.3 Andnroid Power Mangement Architecture with wake locks.

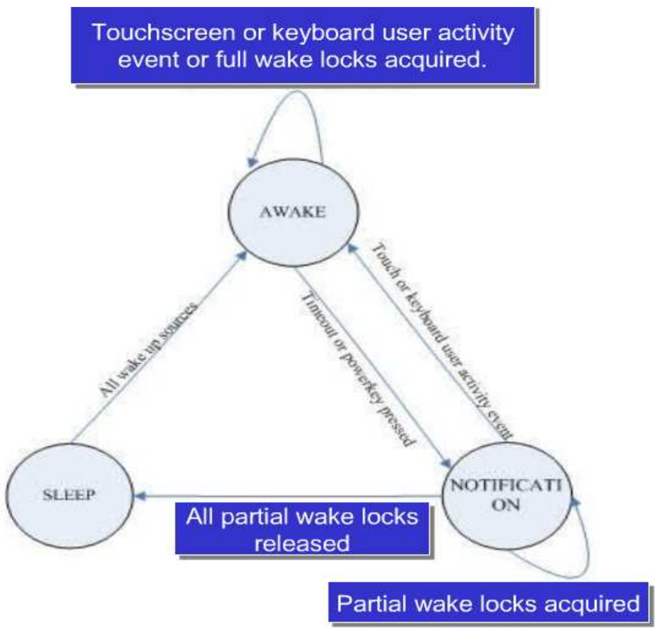

Touchscreen or keyboard user activity event or full wake locks acquired.
AWAKE
Touch or keyboard user activity event
Timeout or powerkey pressed
All wake up sources
SLEEP
NOTIFICATION
All partial wake locks released
Partial wake locks acquired

Figure 3.4 A finite state machine of Android PM.

Figure 3.4 shows that the full state machine. There are three states: “SLEEP”, “NOTIFICATION”, and “AWAKE”. The scenario is: *While a user application acquire full wake lock or touch screen/keyboard activity event, the machine will enter or keep in the “AWAKE”. If timeout happen or power key pressing, the machine will enter “NOTIFICATION”. While partial wake locks acquiring, it will keep in “NOTIFICATION”. While all partial locks released, the machine will go into “SLEEP”. In “SLEEP” mode, it will transit if all resource awake.* This state machine make power saving of Android more feasible for mobile devices.

Finally, the main concept of Android PM is through wake locks and time out mechanism to switch state of system power, so that system power consumption will decrease. The Android PM Framework provides a software solution to accomplish power saving for mobile devices. The following diagram (Figure 3.5) shows the overall architecture of Android PM.

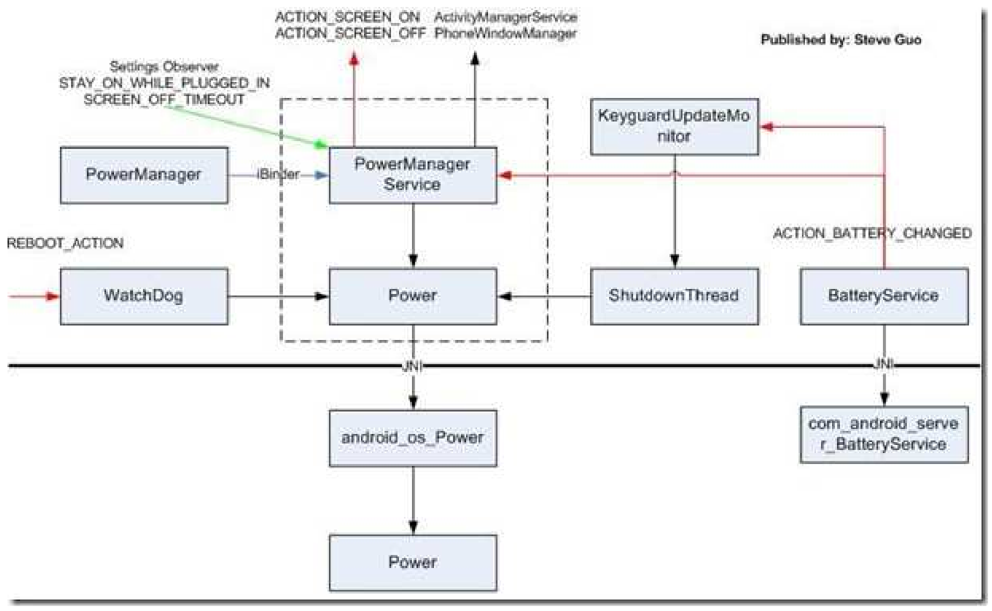


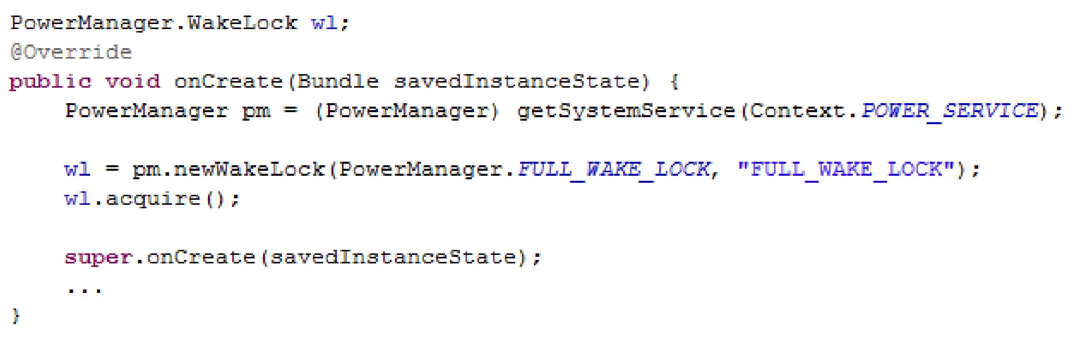
Figure 3.5 The overall architecture of Android PM.

4. Android PM Implementation

Android PM Framework provides a service for user space applications through the class PowerManger to achieve power saving. Hence, the app must get an instance of PowerManger in order to enforce its power requirements. The flow of exploring Wake locks are here:

1. ***Acquire handle to the PowerManager service by calling Context.getSystemService().***
2. ***Create a wake lock and specify the power management flags for screen, timeout, etc.***
3. ***Acquire wake lock.***
4. ***Perform operation such as play MP3.***
5. ***Release wake lock.***

Here we provide an example code of PM. We will put the wake locks code on the function of onCreate() which will initialize first while the program start. And then release locks on the function of onDestroy() method. Then, we can control different type wake locks to accept different timeout or power saving mechanisms after finishing the implement.



```
PowerManager.WakeLock wl;
@Override
public void onCreate(Bundle savedInstanceState) {
    PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);

    wl = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "FULL_WAKE_LOCK");
    wl.acquire();

    super.onCreate(savedInstanceState);
    ...
}
```

5. Conclusion

Power saving is an important issue for mobile devices, and there are many ways to implement. How to design a PM for mobile device need is a good question. Android builds up its user space level solution in order to maintain Linux fundamental support and increase flexibilities. Android PM already supports some power saving type for modern diverse embedded systems.

The number of wake locks type might be not enough for diverse power consuming I/O devices. For example, WiFi antenna is a main power consumption device. Android doesn't automatically turn off WiFi if user is not using.

There are two main points that we think PM can be improved. First, Nowadays, CPU can enter into more states for power saving and usability purpose. There could be more types of wake lock to set the power saving mode specifically.

Second, the old PM system usually sense the keyboard or touch screen activity to judge whether should enter power saving mode or not. We purpose a new concept that sensors can be used to shorten the length of device timeout. For example, if the mobile device has light sensor, we can use it to tune the brightness of LCD and keyboard. Furthermore, we can use motion sensor to detect user's behavior. To sum up, PM is very important for mobile device but it still have room for improvements.

References

1. Robin Kravets, P. Krishnan, Power management techniques for mobile communication, international Conference on Mobile Computing and Networking.
2. Dynamic power management for embedded systems IBM and MontaVista Software Version 1.1, November 19, 2002

3. Robin Kravets , P. Krishnan , Application-driven power management for mobile communication.
4. Andreas Weissel, Frank Bellosa, Process cruise control: event-driven clock scaling for dynamic power management.
5. APM V1.2 spec.
6. ACPI, http://www.lesswatts.org/projects/acpi/index.php
7. Android Project - Power Management, http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html
8. Steve Guo, Android Power Management
9. Matt Hsu, Jim Huang, Power Management from Linux Kernel to Android, 0xlab, 2009.