ACPI

chuang.dong@windriver.com

1.ACPI Concept

1.1 Concept

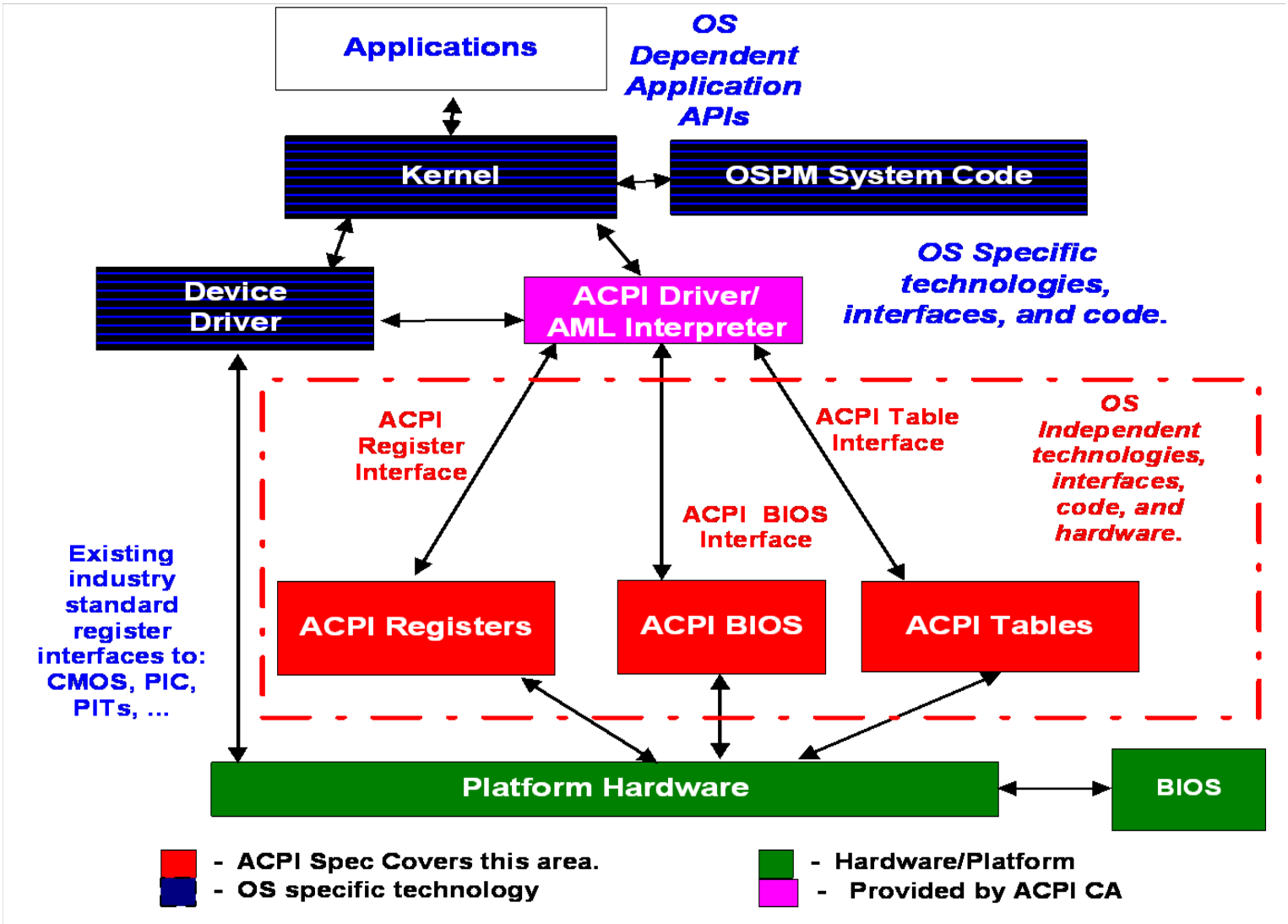
ACPI(Advanced Configuration and Power interface) is an open industry specification co-developed by Hewlett-Packed,Intel,Microsoft,Phoenix and Toshiba.

1.2 Purpose

ACPI establishes industry-standard interface enabling OS-directed configuration,power management,and thermal management of mobile,desktop,and server platforms.

http://www.acpi.info

1.3 ACPI Structure



2 AML Concept

2.1 Concept

AML is the ACPI Control Method virtual machine language,machine code for a virtual machine that is supported by an ACPI-compatible OS.ACPI control methods can be written in AML,but humans ordinarily write control methods in ASL.

2.2 Purpose

OS independent technologies,interfaces,code and hardware.

2.3 AML

AML code was supplied by BIOS,OS has a AML interpreter to interpret AML code and run it.

2.3.1 ACPI Register Interface

Describe ACPI relative registers. OS could operate devices by these registers,therefore,to get info from them.

2.3.2 AML BIOS Interface

Describe ACPI relative registers with BIOS

2.3.3 AML Table Interface

These table describe hardware feature,such as: RSDP,SDTH,RSDT,FADT,DSDT,etc.the important table for us is DSDT.

2.4 ACPI Table

2.4.1 Advantage

1) BIOS will download the ACPI Table into RAM from BIOS ROM,then left some info to OS in order to search easily.

2) Take RSDP Table as an example,BIOS will put it into the address which located in E0000h~FFFFh,then OS can search it easily through signature way to get ACPI Table entry point.

3) OS need to have a AML Translator to translate these AML code,after that run them.

4) As it is different about the registers of host controller in every platform, thus,BIOS need to show these info for OS through ACPI Table,where is the register in.It is great for OS,because of the reason that OS doesn't care what the Platform is,who only focus on the HW info getting from BIOS.

2.4.2 ACPI Function

1) System power management

2) Device power management

3) Processor power management

4) Device and processor performance management

5) Configurration/Plug and Play

6) System event

7) Battery management

8) Embedded Controller

9) SMBus Controller

2.4.3 Example

```
Device (EHC0)
{
 Name (_ADR, 0x001D0007) // _ADR: Address
 Name (_S4D, 0x02) // _S4D: S4 Device State
 Name (_S3D, 0x02) // _S3D: S3 Device State----------------------> set up S3 Sleep
 Name (_S2D, 0x02) // _S2D: S2 Device State
 Name (_S1D, 0x02) // _S1D: S1 Device State
.....................................
 Method (_PRW, 0, NotSerialized) // _PRW: Power Resources for Wake
 {
 Return (GPRW (0x0D, 0x04)) ----------------------------------------> can be waked up in S4 Sleep.
 }
}
```

If the device can wake the system from the S3 system sleeping state (see _PRW) then the device must support wake in the D-state returned by this object. However, OSPM cannot assume wake from the S3 system sleeping state is supported in any lower D-state unless specified by a corresponding _S3W object. The table below provides a mapping from Desired Actions to Resultant D-state entered based on the values returned from the _S3D, _PRW, and _S3W objects if they exist . (D/C means Don't Care – evaluation is irrelevant, and N/A means Non Applicable – object does not exist).

Table 7-202 S3 Action / Result Table

Desired Action	_S3D	_PRW	_S3W	Resultant D-state
Enter S3	N/A	D/C	N/A	OSPM decides
Enter S3, No Wake	2	D/C	D/C	Enter D2 or D3
Enter S3, Wake	2	3	N/A	Enter D2
Enter S3, Wake	2	3	3	Enter D2 or D3
Enter S3, Wake	N/A	3	2	Enter D0, D1 or D2

19.5.86 Name (Declare Named Object)

Syntax

Name (*ObjectName, Object***)**

Arguments

Creates a new object named *ObjectName.* Attaches *Object* to *ObjectName* in the Global ACPI namespace.

Description

Creates *ObjectName* in the namespace, which references the *Object*.

Example

The following example creates the name PTTX in the root of the namespace that references a package.

```
Name (\PTTX,                    // Port to Port Translate Table
    Package () {Package () {0x43, 0x59}, Package) {0x90, 0xFF}}
)
```

The following example creates the name CNT in the root of the namespace that references an integer data object with the value 5.

```
Name (\CNT, 5)
```

2.4.4 Example

This is a segment of DSDT Table.These sentences below are prompted that this device support S3,S4,S5.

```
If (SS3)
{
   Name (_S3, Package (0x04)  // _S3_: S3 System State
   {
      0x05,
```

```
        Zero,
        Zero,
        Zero
    })
}

If (SS4)
{
    Name (_S4, Package (0x04)  // _S4_: S4 System State
    {
        0x06,
        Zero,
        Zero,
        Zero
    })
}

Name (_S5, Package (0x04)  // _S5_: S5 System State
{
    0x07,
    Zero,
    Zero,
    Zero
})
```

2.4.5 Question:

What's the meaning of S3,S4,S5?

2.4.6 Answer:

S1 = “standby”(stop processor and keep power on to everything)

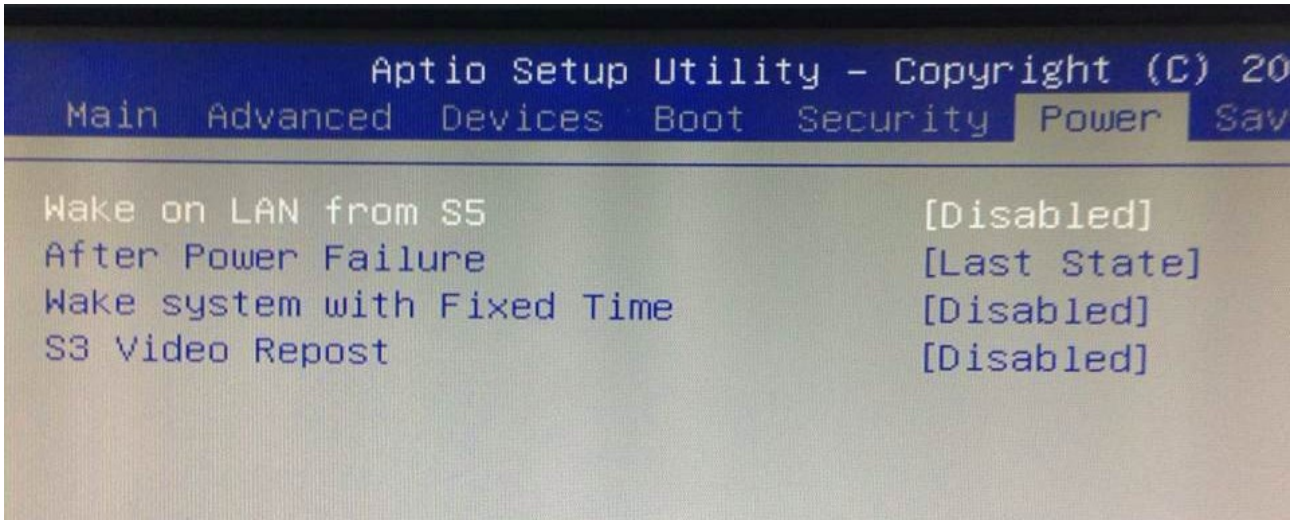
S2(nerver implemented, seen as the same as S3)

S3 = “mem”(save and restore processor state via memory,keeping some things powered-suspend)

S4 = “disk”(save and restore processor state via disk,keeping nothing powered-suspend)

S5 = poweroff,no state saved.

2.4.7 BIOS Power management



2.5 How to Debug ACPI Table

2.5.1 Install necessary tools

1) Install acpidump

```
sudo apt-get install acpidump
sudo apt-get install iasl
```

2) Export DSDT Table

```
acpidump --table DSDT > dsdt.dat
```

3) Disassembling dsdt.dat

```
iasl dsdt.dat
```

then there is a dsdt.dsl file.

4) Open dsdt.dsl file,Check the key code:

For example to S3 Sleep,Check the key code:"_S3D" and "_S3", then compare the result with ACPI_5_Errata.pdf.

5) Check the format of DSDT table

```
dmesg > boot_info.txt
vi boot_info.txt & find DSDT
[    0.000000] ACPI: DSDT 0x00000000BBD0F018 007D91 (v01 IBM    RSS      00000000 INTL
```

20051117)

like this, it is Linux format file.

2.5.2 Modify the DSDT table

1) modify dsdt.dsl

2) compile it:iasl -tc dsdt.dsl (then created a file named:dsdt.hex)

3) cp dsdt.hex SRC/include

4) make menuconfig or modify .config file

5) CONFIG_ACPI_CUSTOM_DSDT_FILE="dsdt.hex"

6) make image

3. Linux power management

3.1 System Sleep status

S1 = “standby”(stop processor and keep power on to everything)

S2(nerver implemented, seen as the same as S3)

S3 = “mem”(save and restore processor state via memory,keeping some things powered-suspend)

S4 = “disk”(save and restore processor state via disk,keeping nothing powered-suspend)

S5 = poweroff,no state saved.

3.1.1 Power command

```
root@128:~# cat /sys/power/state
freeze mem disk
root@128:~#
```

3.1.2 ACPI command

```
root@128:~# cat /proc/acpi/wakeup
Device  S-state   Status   Sysfs node
P0P2      S4    *disabled
PEGP      S4    *disabled
P0P1      S4    *disabled  pci:0000:00:1e.0
PS2K      S3    *enabled   pnp:00:06
PS2M      S3    *disabled
USB0      S3    *enabled   pci:0000:00:1d.0
USB1      S3    *enabled   pci:0000:00:1d.1
USB2      S3    *enabled   pci:0000:00:1d.2
USBR      S4    *disabled
EHC0      S4    *enabled   pci:0000:00:1d.7
USB4      S4    *enabled   pci:0000:00:1a.1
USB5      S4    *disabled
EHC1      S3    *enabled   pci:0000:00:1a.7
HDEF      S4    *disabled  pci:0000:00:1b.0
RP01      S4    *disabled
PXSX      S4    *disabled
RP02      S4    *disabled
```

```
PXSX      S4    *disabled
RP03      S4    *disabled
PXSX      S4    *disabled
RP04      S4    *disabled
PXSX      S4    *disabled
RP05      S4    *disabled
PXSX      S4    *disabled
RP06      S4    *disabled
PXSX      S4    *disabled
GLAN      S4    *enabled   pci:0000:00:19.0
PWRB      S3    *enabled   platform:PNP0C0C:00
root@128:~#
```

3.1.3 Usable script

How to clear which USB port connect which USB device:

```
sh-4.3# grep  "USB.*pci" /proc/acpi/wakeup | cut -d ':' -f 2- | while read aaa; do find /dev/ .udev
-name "*$aaa*" -print -exec grep "$aaa" /proc/acpi/wakeup \; -exec echo \; ; done
find: '.udev': No such file or directory
/dev/input/by-path/pci-0000:00:1d.0-usb-0:2:1.0-event-mouse
USB1      S3    *enabled   pci:0000:00:1d.0

/dev/input/by-path/pci-0000:00:1d.0-usb-0:1:1.0-event-mouse
USB1      S3    *enabled   pci:0000:00:1d.0

/dev/input/by-path/pci-0000:00:1d.0-usb-0:2:1.0-mouse
USB1      S3    *enabled   pci:0000:00:1d.0

/dev/input/by-path/pci-0000:00:1d.0-usb-0:1:1.0-mouse
USB1      S3    *enabled   pci:0000:00:1d.0

find: '.udev': No such file or directory
find: '.udev': No such file or directory
sh-4.3#
```

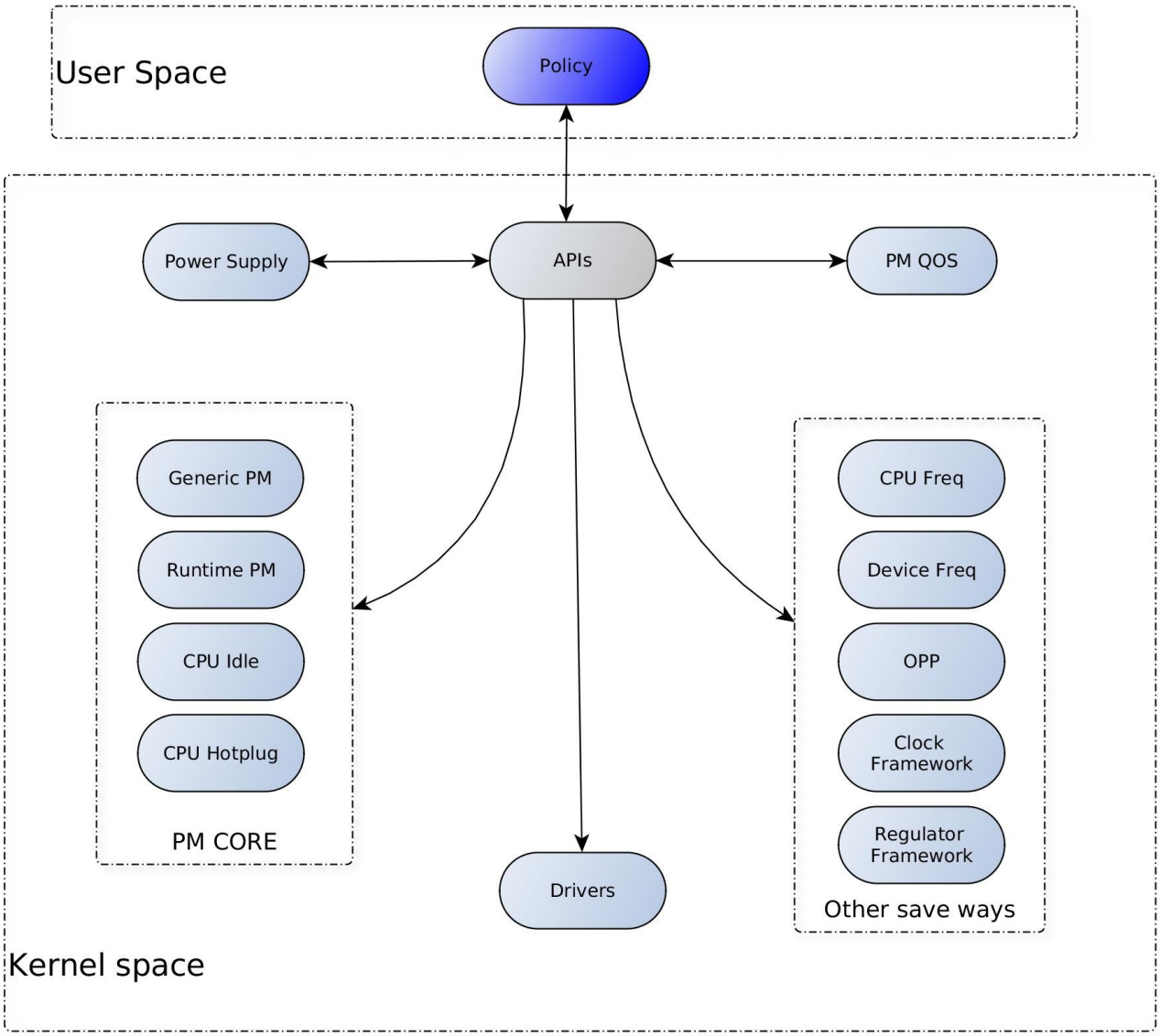
3.1.4 How to run power management

```
echo “mem” > /sys/power/state
echo “freeze” > /sys/power/state
echo “disk” > /sys/power/state
```

Freeze: frozen processes + suspended devices + idle processors

3.2 Linux Power management policy

This is the Linux power management structure.



3.2.1 Important terms

1. Power Supply: It is a framework of battery&charger driver.In order to monitor the status of power supply for user.
2. Clock Framework: It is a framework of clock driver. In charge of system clock resource.

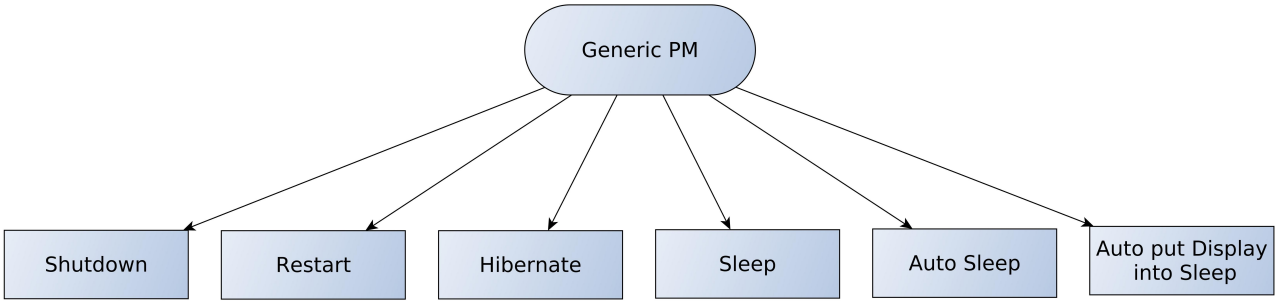
3. Regulator Frame work: It is a framework of voltage/current regulator. In order to operate different module’s voltage and current.
4. CPU Idle: It is a framework of CPU Idle status.
5. Generic PM: It is a traditional power management.Such as Power off,Suspend to RAM,Suspend to Disk,Hibernate,etc.
6. Runtime PM: Running power management.Auto manage the saving status through kernel,instant of user’s operation.
7. CPU Freq: It is a framework of operating CPU frequency.
8. Device Freq: It is a framework of operating device frequency.
9. OPP(Operating Performance Point): It is a different group of SOCs and Device,which can operate voltage and current cooperatively.
10. PM Qos: It is a kind of quality under special running status,include different voltage,frequency,switching module,etc.The analyze param include latency,timeout and throughput,The unite is us, us and kb/s.Through analyze these param to get a balance status for system.

3.2.2 Linux Power source code

1. kernel/power/ *
2. drivers/power/
3. drivers/base/power/*
4. drivers/cpuidle/*
5. drivers/cpufreq/*
6. drivers/devfreq/*
7. include/linux/power_supply.h
8. include/linux/cpuidle.h
9. include/linux/cpufreq.h
10. include/linux/cpu_pm.h
11. include/linux/device.h
12. include/linux/pm.h
13. include/linux/pm domain.h
14. include/linux/pm runtime.h
15. include/linux/pm wakeup.h
16. include/linux/suspend.h
17. Documentation/power/*.txt

3.3 Generic PM

3.3.1 The function structure of Generic PM

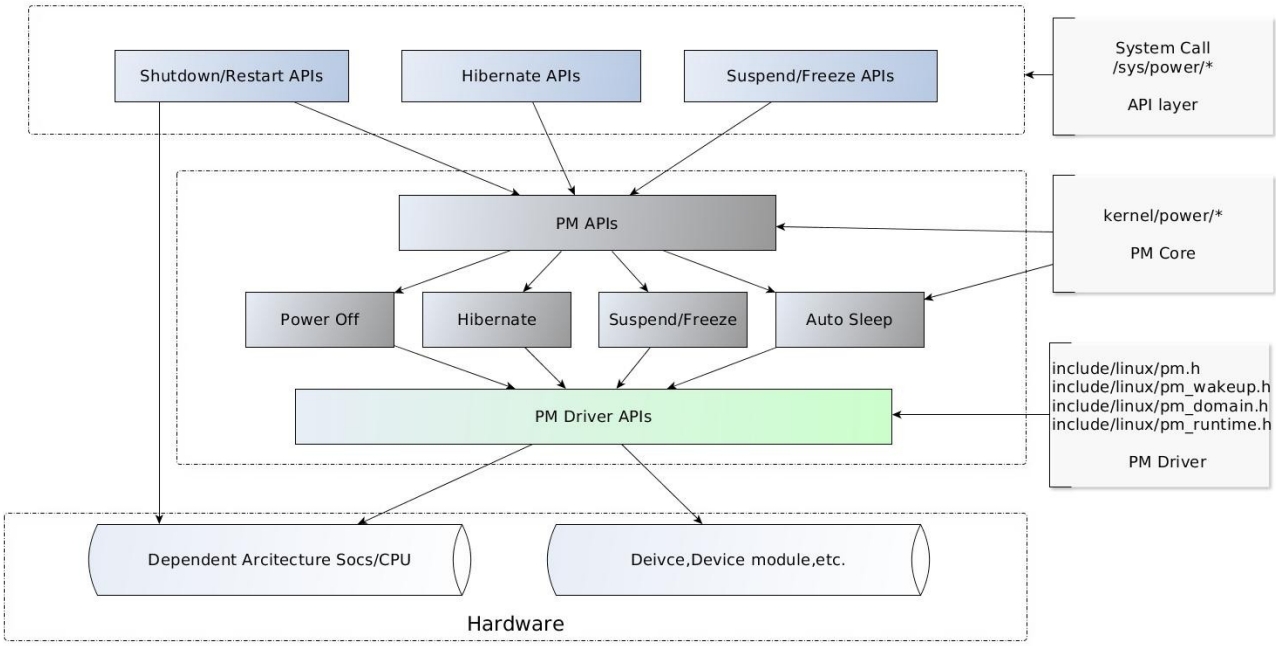


3.3.2 The important terms

1. Shutdown: Power off the computer,Not use it anymore.
2. Restart: Restart computer,Not use it anymore.
3. Hibernate:It means when not use computer.Save the current status,include,running program,opening browser,opening MP3,etc, into ROM,then shutdown the computer,After restarting it next time,the saving status recover again.
4. Sleep: When not use computer,save the current status into RAM,the power supply continually. When the wake up event happened,recover ROM again.
5. Auto Sleep:In a special status,the computer don’t used,get to sleep.For example,set an Alarm,the computer get to sleep after 1hour or at designed time.
6. Auto put display into Sleep: The principle is the same with Auto Sleep,The target is display.

Key word:Not use computer any more.

3.3.3 The software structure of Generic PM



API Layer:Provide interface to user,in order to operate power management

PM Core:PM logical,It is a abstract layer

PM Driver:Driver abstract layer.Provide uniformed interface for driver.

3.3.4 Key structure for dpm

1)Struct dev_pm_ops (include/linux/pm.h)

```
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);
};
```

The purpose:

Provide uniformed interface for driver.

The Sequence:

The sequence of function called in the process of suspend and resume:

prepare—>suspend—>suspend_late—>suspend_noirq—>wakeup—>resume_noirq—>resume_early—>resume—>complete

What the complexe thins is that Driver Engineers have to clear where and when add this callback function in different driver.

What's the detailed explaination for each elemt of dev_pm_ops,Please refer to the explaination in this file(incluide/linux/pm.h)

2) Struct dev_pm_info (include/linux/pm.h)

```
struct dev_pm_info {
	pm_message_t		power_state;
	unsigned int		can_wakeup:1;
	unsigned int		async_suspend:1;
	bool			is_prepared:1;	/* Owned by the PM core */
	bool			is_suspended:1; /* Ditto */
	bool			is_noirq_suspended:1;
	bool			is_late_suspended:1;
	bool			ignore_children:1;
	bool			early_init:1;	/* Owned by the PM core */
	bool			direct_complete:1;	/* Owned by the PM core */
	spinlock_t		lock;
#ifdef CONFIG_PM_SLEEP
	struct list_head	entry;
	struct completion	completion;
	struct wakeup_source	*wakeup;
	bool			wakeup_path:1;
	bool			syscore:1;
#else
	unsigned int		should_wakeup:1;
#endif
#ifdef CONFIG_PM
	struct timer_list	suspend_timer;
	unsigned long		timer_expires;
	struct work_struct	work;
	wait_queue_head_t	wait_queue;
```

```
	atomic_t		usage_count;
	atomic_t		child_count;
	unsigned int		disable_depth:3;
	unsigned int		idle_notification:1;
	unsigned int		request_pending:1;
	unsigned int		deferred_resume:1;
	unsigned int		run_wake:1;
	unsigned int		runtime_auto:1;
	unsigned int		no_callbacks:1;
	unsigned int		irq_safe:1;
	unsigned int		use_autosuspend:1;
	unsigned int		timer_autosuspends:1;
	unsigned int		memalloc_noio:1;
	enum rpm_request	request;
	enum rpm_status		runtime_status;
	int			runtime_error;
	int			autosuspend_delay;
	unsigned long		last_busy;
	unsigned long		active_jiffies;
	unsigned long		suspended_jiffies;
	unsigned long		accounting_timestamp;
#endif
	struct pm_subsys_data	*subsys_data;  /* Owned by the subsystem. */
	void (*set_latency_tolerance)(struct device *, s32);
	struct dev_pm_qos	*qos;
};
```

dev_pm_info is used for saving PM status,such as wakeup capability, wake up source,wait queue,etc.

3) Struct dev_pm_domain (include/linux/pm.h)

```
struct dev_pm_domain {
	struct dev_pm_ops	ops;
	void (*detach)(struct device *dev, bool power_off);
	int (*activate)(struct device *dev);
	void (*sync)(struct device *dev);
```

```
    void (*dismiss)(struct device *dev);
};
```

Purpose:

dev_pm_domain stand for power domain,it forcus on devcie.As you know,the bus_type,device_driver,class,device_type,etc,all of these data structure stand for device driver,thus,the operation of power management in charged by device driver,it is reasonable.But kernel allow this type of device,which doesn't has driver.For this case,we need dev_pm_domain to deal with this thing.

4) Device callback function in Device module

```
struct bus_type {
        ...
        const struct dev_pm_ops *pm;
        ...
 };
 struct device_driver {
        ...
       const struct dev_pm_ops *pm;
       ...
};
struct class {
       ...
       const struct dev_pm_ops *pm;
       ...
};
struct device_type {
       ...
       const struct dev_pm_ops *pm;
};
struct device {
       ...
       struct dev_pm_info      power;
       struct dev_pm_domain    *pm_domain;
       ...
};
```

3.3.5 Key generic PM Device function

```
extern int pm_generic_prepare(struct device *dev);
extern int pm_generic_suspend_late(struct device *dev);
extern int pm_generic_suspend_noirq(struct device *dev);
extern int pm_generic_suspend(struct device *dev);
extern int pm_generic_resume_early(struct device *dev);
extern int pm_generic_resume_noirq(struct device *dev);
extern int pm_generic_resume(struct device *dev);
extern int pm_generic_freeze_noirq(struct device *dev);
extern int pm_generic_freeze_late(struct device *dev);
extern int pm_generic_freeze(struct device *dev);
extern int pm_generic_thaw_noirq(struct device *dev);
extern int pm_generic_thaw_early(struct device *dev);
extern int pm_generic_thaw(struct device *dev);
extern int pm_generic_restore_noirq(struct device *dev);
extern int pm_generic_restore_early(struct device *dev);
extern int pm_generic_restore(struct device *dev);
extern int pm_generic_poweroff_noirq(struct device *dev);
extern int pm_generic_poweroff_late(struct device *dev);
extern int pm_generic_poweroff(struct device *dev);
extern void pm_generic_complete(struct device *dev);
```

Purpose:

In order to operate PM Device callback function.All of these functions encapsulation the seperately PM Device callback function.It is a convenient interface for kernel.

3.3.6 Key PM behavior generic function

```
#ifdef CONFIG_PM_SLEEP
extern void device_pm_lock(void);
extern void dpm_resume_start(pm_message_t state);
extern void dpm_resume_end(pm_message_t state);
extern void dpm_resume_noirq(pm_message_t state);
extern void dpm_resume_early(pm_message_t state);
extern void dpm_resume(pm_message_t state);
extern void dpm_complete(pm_message_t state);
```

```
extern void device_pm_unlock(void);
extern int dpm_suspend_end(pm_message_t state);
extern int dpm_suspend_start(pm_message_t state);
extern int dpm_suspend_noirq(pm_message_t state);
extern int dpm_suspend_late(pm_message_t state);
extern int dpm_suspend(pm_message_t state);
extern int dpm_prepare(pm_message_t state);

extern void __suspend_report_result(const char *function, void *fn, int ret);

#define suspend_report_result(fn, ret)                  \
        do {                                            \
                __suspend_report_result(__func__, fn, ret);             \
        } while (0)

extern int device_pm_wait_for_dev(struct device *sub, struct device *dev);
extern void dpm_for_each_dev(void *data, void (*fn)(struct device *, void *));
```

Purpose:

These function separately package indepentent function for calling conveniently.

For example:

```
int dpm_prepare(pm_message_t state)
{
    …...
    while (!list_empty(&dpm_list)) {
        …..
        device_prepare(dev, state);
        …..
        dev->power.is_prepared = true;
        list_move_tail(&dev->power.entry, &dpm_prepared_list);
        …..
        }
}
static int device_prepare(struct device *dev, pm_message_t state)
```

```
{
   …...
         if (dev->pm_domain) {
            info = "preparing power domain ";
            callback = dev->pm_domain->ops.prepare;
      } else if (dev->type && dev->type->pm) {
            info = "preparing type ";
            callback = dev->type->pm->prepare;
      } else if (dev->class && dev->class->pm) {
            info = "preparing class ";
            callback = dev->class->pm->prepare;
      } else if (dev->bus && dev->bus->pm) {
            info = "preparing bus ";
            callback = dev->bus->pm->prepare;
      }

      if (!callback && dev->driver && dev->driver->pm) {
            info = "preparing driver ";
            callback = dev->driver->pm->prepare;
      }

      if (callback) {
            trace_device_pm_callback_start(dev, info, state.event);
            ret = callback(dev);
            trace_device_pm_callback_end(dev, ret);
      }
   …..
}
```

Question:

There is a dpm_list,which link the can.sleep device up.So what's time to add them?

Answer:

device_add--->device_pm_add

Other list:

```
LIST_HEAD(dpm_list);
static LIST_HEAD(dpm_prepared_list);
```

```
static LIST_HEAD(dpm_suspended_list);
static LIST_HEAD(dpm_late_early_list);
static LIST_HEAD(dpm_noirq_list);
```

Sequence:

Select which PM Device callback function,depend on the below function sequence.

dev->pm_domain->ops、dev->type->pm、dev->class->pm、dev->bus->pm、dev->driver->pm

Others:

dpm_suspend-->suspend() callback(s)

dpm_suspend_start = dpm_prepare + dpm_suspend

dpm_suspend_end-->suspend_late() callback(s) + ->suspend_noirq() callback(s)

dpm_resume、dpm_complete、dpm_resume_start、dpm_resume_end:For wake up

3.3.7 Key structure for pm

```
struct platform_suspend_ops {
	int (*valid)(suspend_state_t state);
	int (*begin)(suspend_state_t state);
	int (*prepare)(void);
	int (*prepare_late)(void);
	int (*enter)(suspend_state_t state);
	void (*wake)(void);
	void (*finish)(void);
	bool (*suspend_again)(void);
	void (*end)(void);
	void (*recover)(void);
};
struct platform_freeze_ops {
	int (*begin)(void);
	int (*prepare)(void);
	void (*restore)(void);
	void (*end)(void);
};
struct syscore_ops {
	struct list_head node;
	int (*suspend)(void);
	void (*resume)(void);
	void (*shutdown)(void);
```

```
};
```

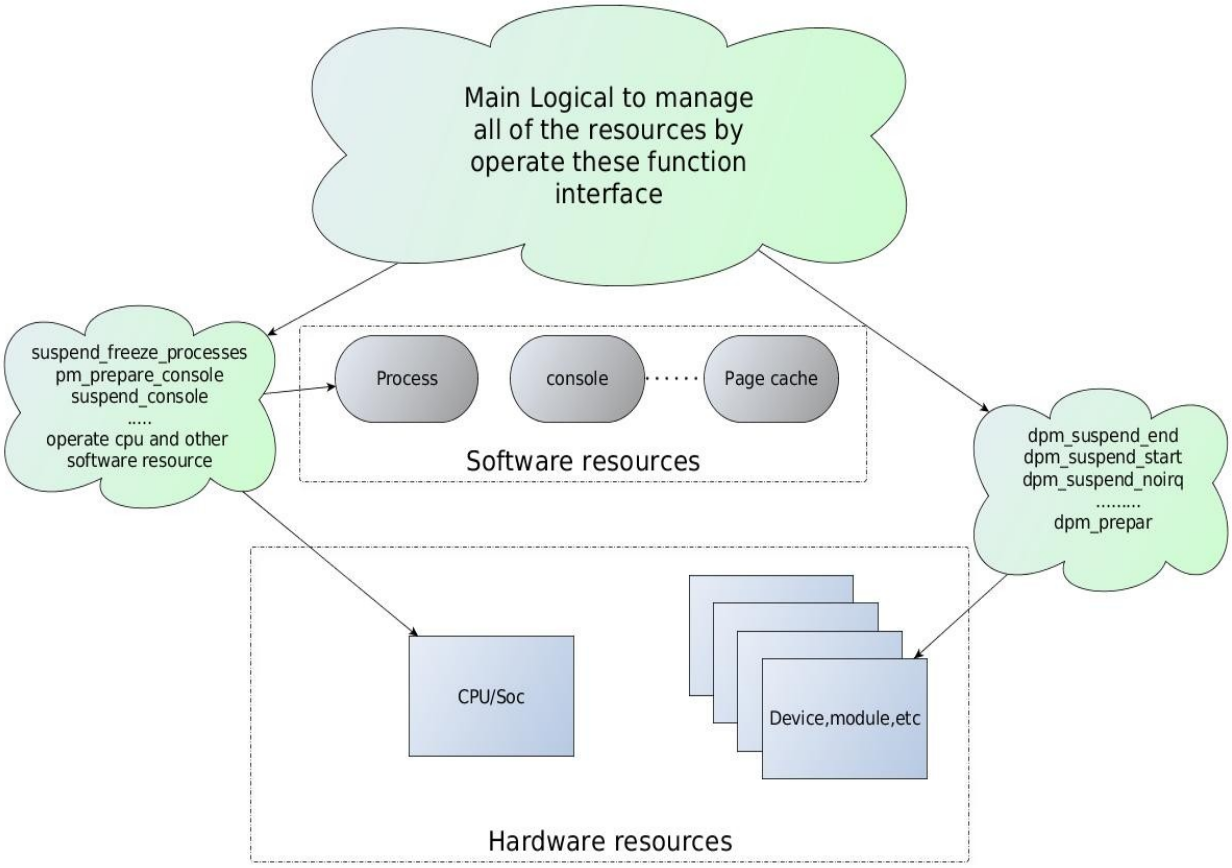
Purpose:

Provide suspend and hibernate structure for PM.In order to provide convenient callback function.

The detailed explanation,please refer to the explanation in kernel(include/linux/suspend.h)

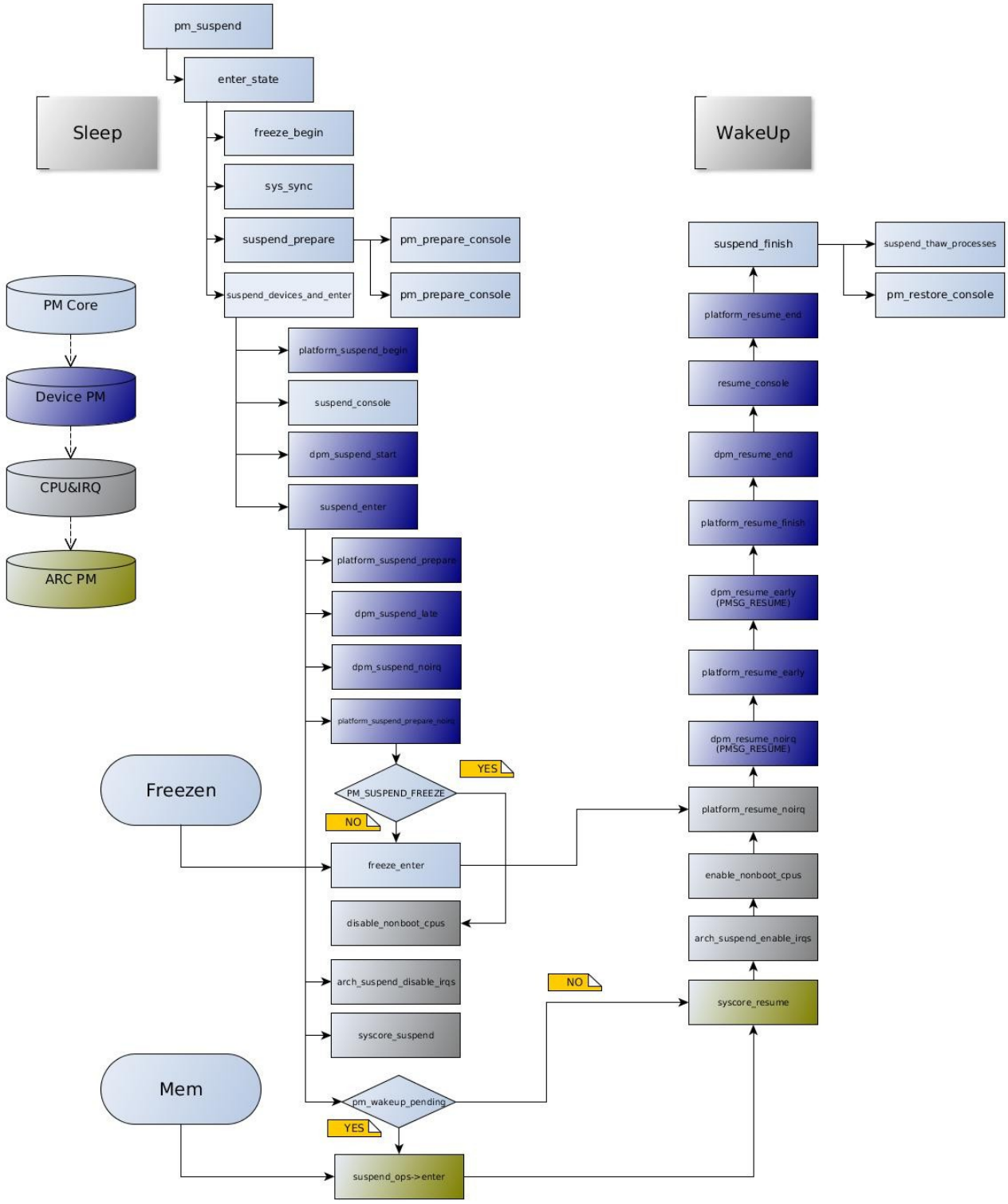
3.3.8 sum up

To sum up the generic PM management.



3.4 Sleep(Suspend)

3.4.1 The Sleep(Suspend)&Wakeup(Resume) Flow Chart:



3.4.2 Sleep(Suspend) Code

1) PM Core Code

```
kernel/power/main.c----user interface(/sys/power/state)
kernel/power/suspend.c----Mainly suspend logical
kernel/power/suspend_test.c----Suspend test logical
kernel/power/console.c----Suspend console logical
kernel/power/process.c----Suspend process logical
```

2) Device Code

```
drivers/base/power/* Please refer to 3.3.4~3.3.6
```

3) Platform dependent PM Code

```
include/linux/suspend.h----Platform dependent PM functions group
arch/xxx/mach-xxx/xxx.c----Platform PM function
arch/xxx/plat-xxx/xxx.c----Platform PM function
```

3.4.3 Wake up source

Question:

Does any terminal can wake computer up?

Answer:

Exactlly,NO,Take a look at the below configuration from USB-port Keyboard

```
wrsadmin@pek-cdong-u14:~$ lsusb -v
….....
Bus 002 Device 015: ID 413c:2107 Dell Computer Corp.
Device Descriptor:
  bLength 18
  bDescriptorType 1
  bcdUSB 2.00
  bDeviceClass 0 (Defined at Interface level)
  bDeviceSubClass 0
  bDeviceProtocol 0
  bMaxPacketSize0 8
  idVendor 0x413c Dell Computer Corp.
```

```
idProduct 0x2107
bcdDevice 1.04
iManufacturer 1 DELL
iProduct 2 Dell USB Entry Keyboard
iSerial 0
bNumConfigurations 1
Configuration Descriptor:
  bLength 9
  bDescriptorType 2
  wTotalLength 34
  bNumInterfaces 1
  bConfigurationValue 1
  iConfiguration 0
  bmAttributes 0xa0
    (Bus Powered)
    Remote Wakeup ----------->This is the key word
  MaxPower 100mA
  Interface Descriptor:
  .....
```

This highlight phrases show that this keyboard has remote wakeup capibility.

3.4.4 Important Function

```
state_store
        pm_suspend
                enter_state
                        freeze_begin
                        sys_sync
                        suspend_prepare
                        suspend_devices_and_enter
                                platform_suspend_begin
                                suspend_console
                                dpm_suspend_start
```

```
                                suspend_enter
                                        platform_suspend_prepare
                                        dpm_suspend_late
                                        dpm_suspend_noirq
                                        platform_suspend_prepare_noirq
                                        disable_nonboot_cpus
                                        arch_suspend_disable_irqs
                                        syscore_suspend
                                        suspend_ops->enter
                                        syscore_resume
                                        arch_suspend_enable_irqs
                                        enable_nonboot_cpus
                                        platform_resume_noirq
                                dpm_resume_noirq(PMSG_RESUME)
                                platform_resume_early
                        dpm_resume_early(PMSG_RESUME)
                        platform_resume_finish
                        dpm_resume_end
                        resume_console
                        platform_resume_end
                suspend_finish
```

state_store

```
static ssize_t state_store(struct kobject *kobj, struct kobj_attribute *attr,
                           const char *buf, size_t n)
{
        suspend_state_t state;
        int error;

        error = pm_autosleep_lock();
        if (error)
                return error;

        if (pm_autosleep_state() > PM_SUSPEND_ON) {
                error = -EBUSY;
                goto out;
```

```
        }

        state = decode_state(buf, n);
        if (state < PM_SUSPEND_MAX)
                error = pm_suspend(state);
        else if (state == PM_SUSPEND_MAX)
                error = hibernate();
        else
                error = -EINVAL;

 out:
        pm_autosleep_unlock();
        return error ? error : n;
}

#define PM_SUSPEND_ON          ((__force suspend_state_t) 0)
#define PM_SUSPEND_FREEZE      ((__force suspend_state_t) 1)
#define PM_SUSPEND_STANDBY     ((__force suspend_state_t) 2)
#define PM_SUSPEND_MEM         ((__force suspend_state_t) 3)
#define PM_SUSPEND_MIN         PM_SUSPEND_FREEZE
#define PM_SUSPEND_MAX         ((__force suspend_state_t) 4)
```

Explanation:

Compare “state” with PM_SUSPEND_MAX, to decide which operation will do,Suspend or Hibernate.

1. pm_suspend:Complete suspend and freeze operation.
2. hibernate:Complete hibernate operation.

pm_suspend

```
int pm_suspend(suspend_state_t state)
{
        int error;

        if (state <= PM_SUSPEND_ON || state >= PM_SUSPEND_MAX)
                return -EINVAL;
```

```
	error = enter_state(state);
	if (error) {
		suspend_stats.fail++;
		dpm_save_failed_errno(error);
	} else {
		suspend_stats.success++;
	}
	return error;
}
```

Explanation:

1. pm_suspend ->enter_state. Judge key variable and update statistics.

enter_state

```
static int enter_state(suspend_state_t state)
{
	int error;

	trace_suspend_resume(TPS("suspend_enter"), state, true);
	if (state == PM_SUSPEND_FREEZE) {
#ifdef CONFIG_PM_DEBUG
		if (pm_test_level != TEST_NONE && pm_test_level <= TEST_CPUS) {
			pr_warning("PM: Unsupported test mode for freeze state,"
				"please choose none/freezer/devices/platform.\n");
			return -EAGAIN;
		}
#endif
	} else if (!valid_state(state)) {
		return -EINVAL;
	}
	if (!mutex_trylock(&pm_mutex))
		return -EBUSY;

	if (state == PM_SUSPEND_FREEZE)
		freeze_begin();
```

```
        trace_suspend_resume(TPS("sync_filesystems"), 0, true);
        printk(KERN_INFO "PM: Syncing filesystems ... ");
        sys_sync();
        printk("done.\n");
        trace_suspend_resume(TPS("sync_filesystems"), 0, false);

        pr_debug("PM: Preparing system for %s sleep\n", pm_states[state]);
        error = suspend_prepare(state);
        if (error)
                goto Unlock;
        if (suspend_test(TEST_FREEZER))
                goto Finish;

        trace_suspend_resume(TPS("suspend_enter"), state, false);
        pr_debug("PM: Entering %s sleep\n", pm_states[state]);
        pm_restrict_gfp_mask();
        error = suspend_devices_and_enter(state);
        pm_restore_gfp_mask();

 Finish:
        pr_debug("PM: Finishing wakeup.\n");
        suspend_finish();
 Unlock:
        mutex_unlock(&pm_mutex);
        return error;
}
```

fs/sync.c

```
/*
 * Sync everything. We start by waking flusher threads so that most of
 * writeback runs on all devices in parallel. Then we sync all inodes reliably
 * which effectively also waits for all flusher threads to finish doing
 * writeback. At this point all data is on disk so metadata should be stable
 * and we tell filesystems to sync their metadata via ->sync_fs() calls.
 * Finally, we writeout all block devices because some filesystems (e.g. ext2)
```

```
 * just write metadata (such as inodes or bitmaps) to block device page cache
 * and do not sync it on their own in ->sync_fs().
 */
SYSCALL_DEFINE0(sync)
{
	int nowait = 0, wait = 1;

	wakeup_flusher_threads(0, WB_REASON_SYNC);
	iterate_supers(sync_inodes_one_sb, NULL);
	iterate_supers(sync_fs_one_sb, &nowait);
	iterate_supers(sync_fs_one_sb, &wait);
	iterate_bdevs(fdatawrite_one_bdev, NULL);
	iterate_bdevs(fdatawait_one_bdev, NULL);
	if (unlikely(laptop_mode))
		laptop_sync_completion();
	return 0;
}
```

Explanation:

1. freeze_begin:refer to 3.5 Freeze.
2. sys_sync: Sync everything
3. suspend_prepare: Prepare something for suspend,please refer to next key function.
4. suspend_devices_and_enter:The important function. Most of the suspend operation run here,please refer to the next key function

suspend_prepare

```
static int suspend_prepare(suspend_state_t state)
{
	int error;

	if (!sleep_state_supported(state))
		return -EPERM;

	pm_prepare_console();
```

```
	error = pm_notifier_call_chain(PM_SUSPEND_PREPARE);
	if (error)
		goto Finish;

	trace_suspend_resume(TPS("freeze_processes"), 0, true);
	error = suspend_freeze_processes();
	trace_suspend_resume(TPS("freeze_processes"), 0, false);
	if (!error)
		return 0;

	suspend_stats.failed_freeze++;
	dpm_save_failed_step(SUSPEND_FREEZE);
 Finish:
	pm_notifier_call_chain(PM_POST_SUSPEND);
	pm_restore_console();
	return error;
}
```

Explanation:

1. pm_prepare_console: Prepare something for suspending console.Please refer to 3.4.5 Suspend console.
2. pm_notifier_call_chain:PM message chain.Please refer to 3.4.6 PM notifier
3. suspend_freeze_processes:Prepare something for suspending processes.Please refer to 3.5 Freeze
4. dpm_save_failed_step(SUSPEND_FREEZE): Update failed data.
5. pm_notifier_call_chain(PM_POST_SUSPEND): PM message chain.Please refer to 3.4.6 PM notifier
6. pm_restore_console: Resume console.Please refer to 3.4.5 Suspend console.

suspend_devices_and_enter

```
int suspend_devices_and_enter(suspend_state_t state)
{
	int error;
	bool wakeup = false;

	if (!sleep_state_supported(state))
```

```
        return -ENOSYS;

    error = platform_suspend_begin(state);
    if (error)
        goto Close;

    suspend_console();
    suspend_test_start();
    error = dpm_suspend_start(PMSG_SUSPEND);
    if (error) {
        pr_err("PM: Some devices failed to suspend, or early wake event detected\n");
        goto Recover_platform;
    }
    suspend_test_finish("suspend devices");
    if (suspend_test(TEST_DEVICES))
        goto Recover_platform;

    do {
        error = suspend_enter(state, &wakeup);
    } while (!error && !wakeup && platform_suspend_again(state));

Resume_devices:
    suspend_test_start();
    dpm_resume_end(PMSG_RESUME);
    suspend_test_finish("resume devices");
    trace_suspend_resume(TPS("resume_console"), state, true);
    resume_console();
    trace_suspend_resume(TPS("resume_console"), state, false);

Close:
    platform_resume_end(state);
    return error;

Recover_platform:
    platform_recover(state);
```

```
        goto Resume_devices;
}
```

1. platform_suspend_begin:freeze_ops->begin or suspend_ops->begin, In term of the sleep status to make up which callback can be runed.
2. suspend_console:Suspend console.Please referto 3.4.5 Suspend console.
3. suspend_test_start:
4. dpm_prepare_start:Call dpm_suspend.Prepare all non-sysdev devices for system PM transition and execute "suspend"callbacks for them.
5. suspend_enter:Make the system enter the given sleep state.Please refer to next Key function.
6. dpm_resume_end(PMSG_RESUME):Call dpm_resume and dpm_complete.Execute "resume" callbacks for all devices and complete the PM transition of the system.
7. resume_console:Resume console.Please refer to 3.4.5 Suspend console.
8. platform_resume_end:Call freeze_ops->end or suspend_ops->end.Resume platform status.
9. platform_recover:Call suspend_ops->recover.Recover platform status.

suspend_enter

```
static int suspend_enter(suspend_state_t state, bool *wakeup)
{
        int error;

        error = platform_suspend_prepare(state);
        if (error)
                goto Platform_finish;

        error = dpm_suspend_late(PMSG_SUSPEND);
        if (error) {
                printk(KERN_ERR "PM: late suspend of devices failed\n");
                goto Platform_finish;
        }
        error = platform_suspend_prepare_late(state);
        if (error)
                goto Devices_early_resume;

        error = dpm_suspend_noirq(PMSG_SUSPEND);
```

```
if (error) {
        printk(KERN_ERR "PM: noirq suspend of devices failed\n");
        goto Platform_early_resume;
}
error = platform_suspend_prepare_noirq(state);
if (error)
        goto Platform_wake;

if (suspend_test(TEST_PLATFORM))
        goto Platform_wake;

/*
 * PM_SUSPEND_FREEZE equals
 * frozen processes + suspended devices + idle processors.
 * Thus we should invoke freeze_enter() soon after
 * all the devices are suspended.
 */
if (state == PM_SUSPEND_FREEZE) {
        trace_suspend_resume(TPS("machine_suspend"), state, true);
        freeze_enter();
        trace_suspend_resume(TPS("machine_suspend"), state, false);
        goto Platform_wake;
}

error = disable_nonboot_cpus();
if (error || suspend_test(TEST_CPUS))
        goto Enable_cpus;

arch_suspend_disable_irqs();
BUG_ON(!irqs_disabled());

error = syscore_suspend();
if (!error) {
        *wakeup = pm_wakeup_pending();
        if (!(suspend_test(TEST_CORE) || *wakeup)) {
```

```
                trace_suspend_resume(TPS("machine_suspend"),
                        state, true);
                error = suspend_ops->enter(state);
                trace_suspend_resume(TPS("machine_suspend"),
                        state, false);
                events_check_enabled = false;
            }
            syscore_resume();
        }
        arch_suspend_enable_irqs();
        BUG_ON(irqs_disabled());

 Enable_cpus:
        enable_nonboot_cpus();

 Platform_wake:
        platform_resume_noirq(state);
        dpm_resume_noirq(PMSG_RESUME);

 Platform_early_resume:
        platform_resume_early(state);

 Devices_early_resume:
        dpm_resume_early(PMSG_RESUME);

 Platform_finish:
        platform_resume_finish(state);
        return error;
}
```

Explanation:

1. platform_suspend_prepare:Call suspend_ops->prepare(),platform prepare for suspend.
2. dpm_suspend_late:Execute "late suspend" callbacks for all devices.
3. platform_suspend_prepare_late:Freeze operation call freeze_ops->prepare,otherwise return 0.
4. dpm_suspend_noirq:Prevent device drivers from receiving interrupts and call the "noirq"

suspend handlers for all non-sysdev devices.This is importan function for device suspend.The detailed explanation in next Key function.

5. platform_suspend_prepare_noirq:Freeze operation call freeze_ops->prepare,otherwise return 0.
6. freeze_enter:Freeze operation enter.Please refer to 3.5 Freeze.
7. disable_nonboot_cpus:Disable no-boot cpus.Please refer to 3.7 CPU HotPlug.
8. arch_suspend_disable_irqs:Call local_irq_disable,Diable local irq.
9. syscore_suspend:executed with one CPU on-line and disabled interrupts
10. pm_wakeup_pending:Check if power transition in progress should be aborted,It is important function,which refer to wakeup count.The detailed explanation in 3.4.7 Wakeup count.
11. suspend_ops->enter:This is the enter that suspend platform,then system is suspended.The detailed explanation in next key function.
12.

suspend_ops->enter

```
suspend_ops->enter-->acpi_suspend_enter(x86)
        (S3)---->acpi_suspend_lowlevel--->x86_acpi_suspend_lowlevel
                do_suspend_lowlevel(arch/x86/kernel/acpi/wakeup_64.S)
                        x86_acpi_enter_sleep_state--->acpi_hw_sleep_dispatch
                                sleep_functions->legacy_function-->acpi_hw_legacy_sleep
                                        acpi_hw_write_pm1_control
                                                sleep_functions->extended_function
                                                sleep_functions->legacy_function


sleep_functions->legacy_function----->acpi_hw_legacy_sleep
        acpi_hw_write_pm1_control
                acpi_hw_write(xpm1a_control_block)
                        acpi_hw_write
                                acpi_hw_write_port
                                        acpi_hw_validate_io_request
                                                acpi_hw_write(xpm1b_control_block)
                                                        acpi_hw_validate_register
                                                acpi_os_write_memory
```

```
                                                    acpi_hw_write_port
```

Explanation:

This is the relative detailed calling function process.It can be a reference when troubleshoot relative bug.

dpm_suspend_noirq

```
dpm_suspend_noirq
        device_suspend_noirq
                __device_suspend_noirq
                        dpm_run_callback
                                error = cb(dev)--->pci_pm_suspend_noirq
                                                        :
                                                        :
                                                usb_dev_suspend
                                                    usb_suspend
                                                            choose_wakeup
                                                            usb_suspend_both
```

Explanation:

This is an example for dpm_suspend_noirq,because this function is really importan for device suspend.Therefore,I take it as an typical instance.

USB Device Example

Take usb-port(EHCI) keyboard as an example, show the process of suspend&resuem

```
 Firstly,in term of recursion method to deal with Sleep and Resume.


1)Disable Keyboard irq
2)Send msg to keyboard,in order to be waked up.
3)Stop Root Hub port transfering among the bus.
4)Set up device software status.
/devices/pci0000:00/0000:00:1d.0/usb5/5-1/
usb_suspend_both
        usb_suspend_interface
                hid_suspend----------------------------------->Keyboard stop irq
        usb_suspend_device
```

generic_suspend---------->device driver

usb_port_suspend---------->No Root Hub itself, it is a port in Root Hub

usb_enable_remote_wakeup------>send msg to keyboard,tail it you can be waked up,because it is your special function:(Configuration Descriptor: Remote Wakeup)

set_port_feature---------->stop Root Hub port transfering among the bus.

usb_set_device_state---------->set up device status.

Secondly,For UHCI

/devices/pci0000:00/0000:00:1d.0/usb5/

usb_suspend_both

usb_suspend_interface

hub_suspend

usb_suspend_device

generic_suspend

hcd_bus_suspend

hcd->driver->bus_suspend----->uhci_rh_suspend

suspend_rh

usb_set_device_state(rhdev, USB_STATE_SUSPENDED);

generic_resume

hcd_bus_resume

uhci_rh_resume

wakeup_rh

3.4.5 Suspend console

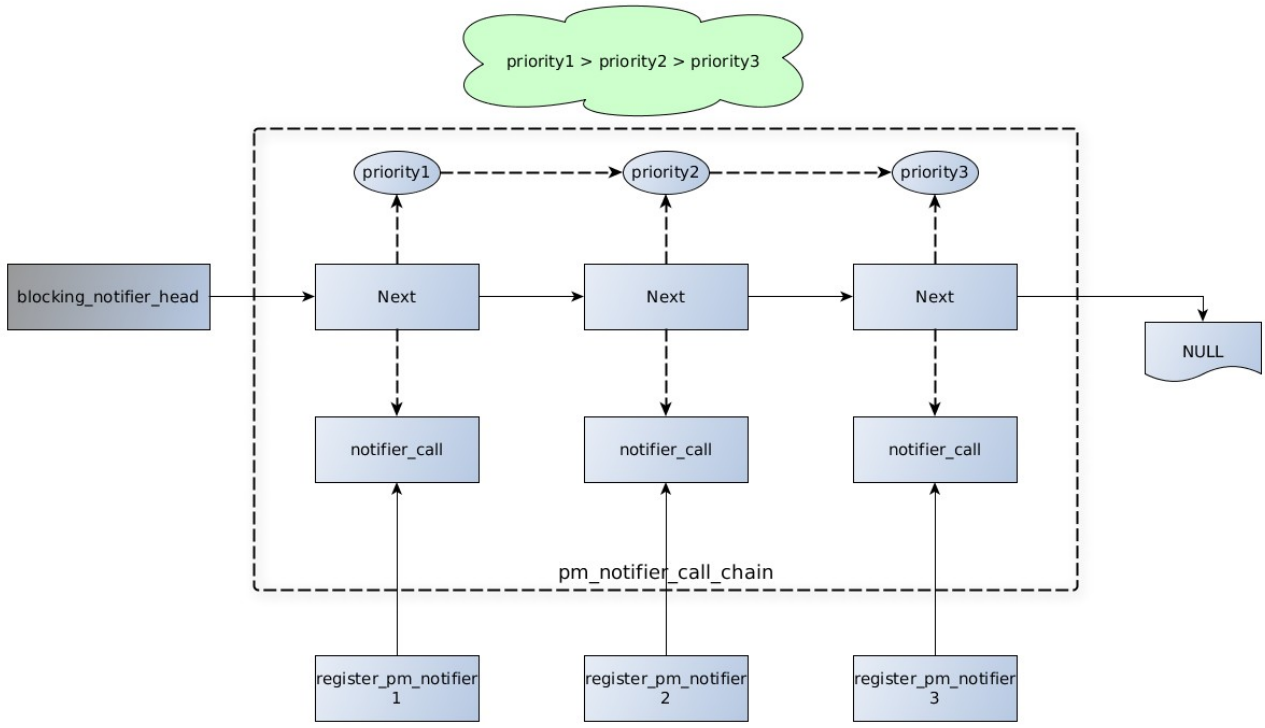
3.4.6 PM notifier

PM notifier base on kernel block notifier chain.The purpose of it is that inform other devices,what status has been changed.In order to allow other devives do relative avtivation.

PM notifier could bypass device module,this function is only for special device.

Key structure & function

```
struct blocking_notifier_head {
        struct rw_semaphore rwsem;
        struct notifier_block __rcu *head;
};
struct notifier_block {
        notifier_fn_t notifier_call;
        struct notifier_block __rcu *next;
        int priority;
};
static BLOCKING_NOTIFIER_HEAD(pm_chain_head);
register_pm_notifier
unregister_pm_notifier
pm_notifier_call_chain
```



3.4.7 Wakeup

Wakeup event

```
struct device {
	...
	struct dev_pm_info	power;
	...
};

struct dev_pm_info {
	…
	unsigned int		can_wakeup:1;
	…s
#ifdef CONFIG_PM_SLEEP
	...
	struct wakeup_source	*wakeup;
	…
#elseif
	unsigned int		should_wakeup:1;
#endif

/**
 * struct wakeup_source - Representation of wakeup sources
 *
 * @total_time: Total time this wakeup source has been active.
 * @max_time: Maximum time this wakeup source has been continuously active.
 * @last_time: Monotonic clock when the wakeup source's was touched last time.
 * @prevent_sleep_time: Total time this source has been preventing autosleep.
 * @event_count: Number of signaled wakeup events.
 * @active_count: Number of times the wakeup source was activated.
 * @relax_count: Number of times the wakeup source was deactivated.
 * @expire_count: Number of times the wakeup source's timeout has expired.
 * @wakeup_count: Number of times the wakeup source might abort suspend.
 * @active: Status of the wakeup source.
 * @has_timeout: The wakeup source has been activated with a timeout.
```

```
 */
struct wakeup_source {
        const char              *name;
        struct list_head        entry;
        spinlock_t              lock;
        struct timer_list       timer;
        unsigned long           timer_expires;
        ktime_t total_time;
        ktime_t max_time;
        ktime_t last_time;
        ktime_t start_prevent_time;
        ktime_t prevent_sleep_time;
        unsigned long           event_count;
        unsigned long           active_count;
        unsigned long           relax_count;
        unsigned long           expire_count;
        unsigned long           wakeup_count;
        bool                    active:1;
        bool                    autosleep_enabled:1;
};
```

Struct dev_pm_info:It records how wake up capibility of device.The detailed info please refer to 3.3.4.

can_wakeup:It determine which device has wakeup capibility.

Struct wakeup_source:If device has wakeup capibility,then all of the wakup info records in this data structure.Each of item of this structure,please check the explanation.

```
extern void wakeup_source_prepare(struct wakeup_source *ws, const char *name);
extern struct wakeup_source *wakeup_source_create(const char *name);
extern void wakeup_source_drop(struct wakeup_source *ws);
extern void wakeup_source_destroy(struct wakeup_source *ws);
extern void wakeup_source_add(struct wakeup_source *ws);
extern void wakeup_source_remove(struct wakeup_source *ws);
extern struct wakeup_source *wakeup_source_register(const char *name);
extern void wakeup_source_unregister(struct wakeup_source *ws);
extern int device_wakeup_enable(struct device *dev);
```

```
extern int device_wakeup_disable(struct device *dev);
extern void device_set_wakeup_capable(struct device *dev, bool capable);
extern int device_init_wakeup(struct device *dev, bool val);
extern int device_set_wakeup_enable(struct device *dev, bool enable);
extern void __pm_stay_awake(struct wakeup_source *ws);
extern void pm_stay_awake(struct device *dev);
extern void __pm_relax(struct wakeup_source *ws);
extern void pm_relax(struct device *dev);
extern void __pm_wakeup_event(struct wakeup_source *ws, unsigned int msec);
extern void pm_wakeup_event(struct device *dev, unsigned int msec);
bool pm_wakeup_pending(void);
```

Key function:

__pm_stay_awake: PM deal with wakeup event,can't be suspended.

__pm_relax: PM has been deal with wakeup event ever.Suspend available.

__pm_wakeup_event: PM deal with wake up event now,after certain seconds. It will be finished.

device_set_wakeup_capable: Set up can_wakeup flag in data structure dev_pm_info.Meanwhile add/delete device power info in sysfs.

pm_wakeup_pending: Determine whether the device suspened is available.

All of these info refer to wakeup source are all observed in relative device power director in sysfs.

Take USB as an example:

```
root@128:/sys/bus/usb/drivers/usb/usb2/power# pwd
/sys/bus/usb/drivers/usb/usb2/power
root@128:/sys/bus/usb/drivers/usb/usb2/power# ls
active_duration       connected_duration   runtime_active_time     runtime_usage       wakeup_active_count  wakeup_max_time_ms
async                 control              runtime_enabled         wakeup              wakeup_count         wakeup_total_time_ms
autosuspend           level                runtime_status          wakeup_abort_count  wakeup_expire_count
autosuspend_delay_ms  runtime_active_kids  runtime_suspended_time  wakeup_active       wakeup_last_time_ms
root@128:/sys/bus/usb/drivers/usb/usb2/power#
```

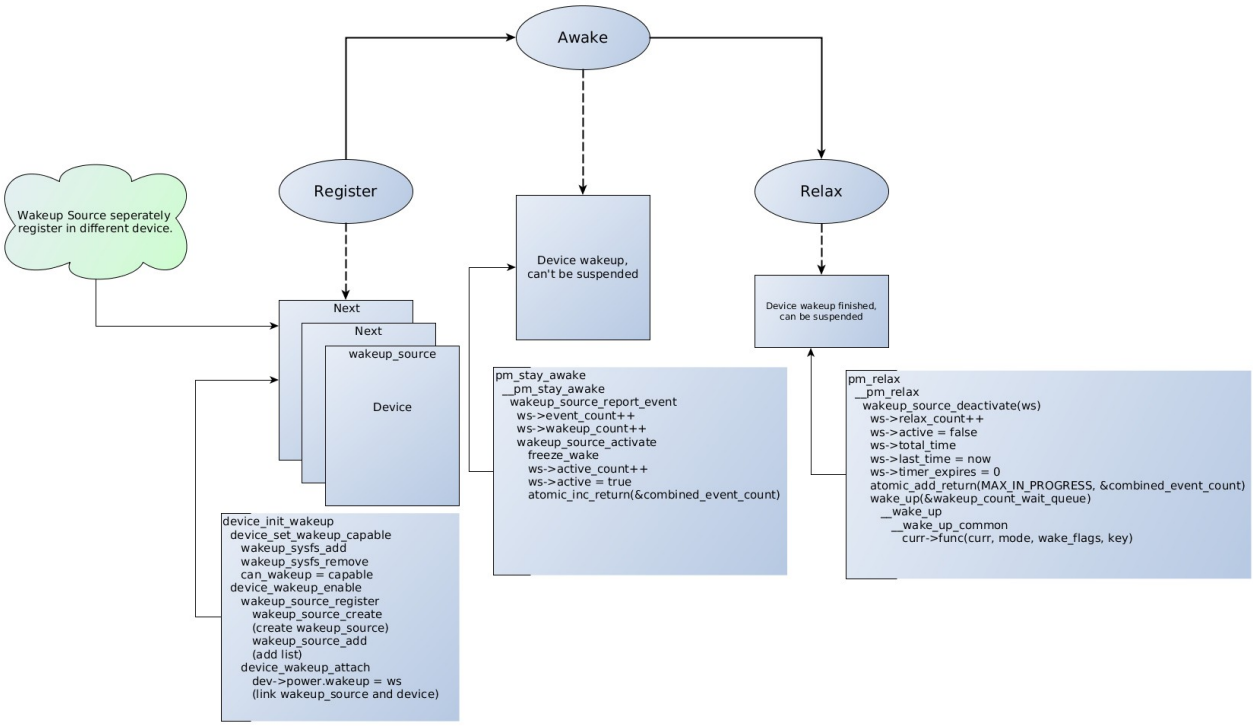
Sum up:

The wakeup function depend on two important points:

1)Whether set can_wakeup flag in data structure dev_pm_info

2)Whether mount wakeup_source point.

The wakeup event status of flow diagram:



Sync(Wakeup count)

Wakeup count is for sync suspend and resume.How to identify this sync? It account for Wakeup count.

The wakeup count could be observed in /sys/power:

```
root@128:/sys/power# ls
disk  image_size  pm_async  pm_freeze_timeout  pm_print_times  pm_test  reserved_size  resume  state  wakeup_count
root@128:/sys/power# cat wakeup_count
0
root@128:/sys/power#
```

The theory of sync:

If a wakeup event occurs when the system is in a sleep state, it simply is woken up. In turn, if an event that would wake the system up from a sleep state occurs when it is undergoing a transition to that sleep state, the transition should be aborted. Moreover, if such an event occurs when the system is in the working state, an attempt to start a transition to the given sleep state should fail during certain period after the detection of the event. Using the 'state' attribute alone is not sufficient to satisfy these requirements, because a wakeup event may occur exactly when 'state' is being written to and may be delivered to user space right before it is frozen, so the event will remain only partially processed until the system is woken up by another event. In particular, it won't cause the transition to a sleep state to be aborted.

This difficulty may be overcome if user space uses 'wakeup_count' before writing to 'state'. It first

should read from 'wakeup_count' and store the read value. Then, after carrying out its own preparations for the system transition to a sleep state, it should write the stored value to 'wakeup_count'. If that fails, at least one wakeup event has occurred since 'wakeup_count' was read and 'state' should not be written to. Otherwise, it is allowed to write to 'state', but the transition will be aborted if there are any wakeup events detected after 'wakeup_count' was written to.

Key global variable:

```
static unsigned int saved_count;
static bool pm_abort_suspend __read_mostly;
```

saved_count: Record wakeup count number.

pm_abour_suspend: Record the suspend action has been aborted.

__read_mostly: Define the variable as a cache date,it means put the variable stay in cache all the time.

Key data structure:

```
struct kobj_attribute {
	struct attribute attr;
	ssize_t (*show)(struct kobject *kobj, struct kobj_attribute *attr,
			char *buf);
	ssize_t (*store)(struct kobject *kobj, struct kobj_attribute *attr,
			 const char *buf, size_t count);
};
#define power_attr(_name) \
static struct kobj_attribute _name##_attr = {	\
	.attr	= {				\
		.name = __stringify(_name),	\
		.mode = 0644,			\
	},					\
	.show	= _name##_show,			\
	.store	= _name##_store,		\
}
```

Key function:

```
bool pm_get_wakeup_count(unsigned int *count, bool block)
bool pm_save_wakeup_count(unsigned int count)
```

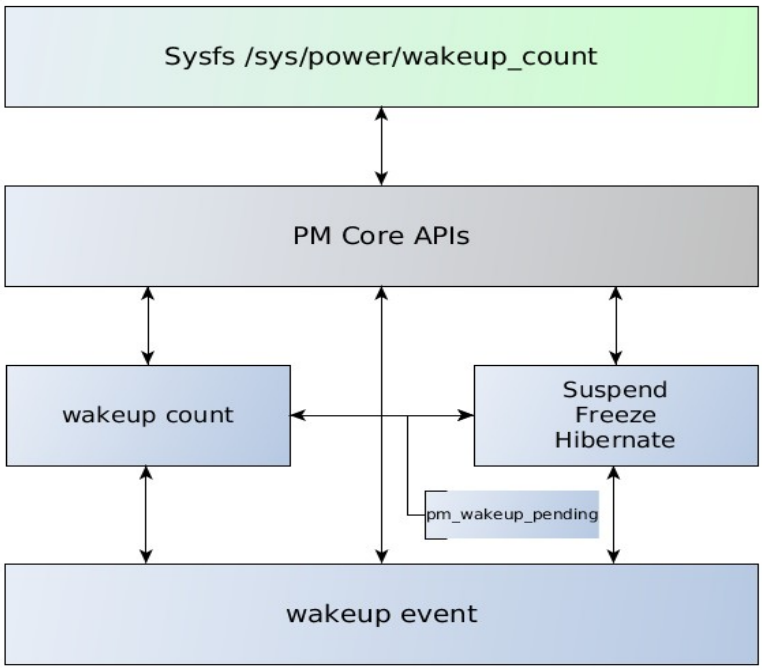
```
bool pm_wakeup_pending(void)
static DECLARE_WAIT_QUEUE_HEAD(wakeup_count_wait_queue);
bool pm_get_wakeup_count(unsigned int *count, bool block)
{
  …
  prepare_to_wait(&wakeup_count_wait_queue, &wait,TASK_INTERRUPTIBLE);
  …
  split_counters(&cnt, &inpr);
  *count = cnt;
  return !inpr;
}
static void split_counters(unsigned int *cnt, unsigned int *inpr)
{
      unsigned int comb = atomic_read(&combined_event_count);

      *cnt = (comb >> IN_PROGRESS_BITS);
      *inpr = comb & MAX_IN_PROGRESS;
}
```

cnt: registered wakeup events

inpr: wakeup events in progress

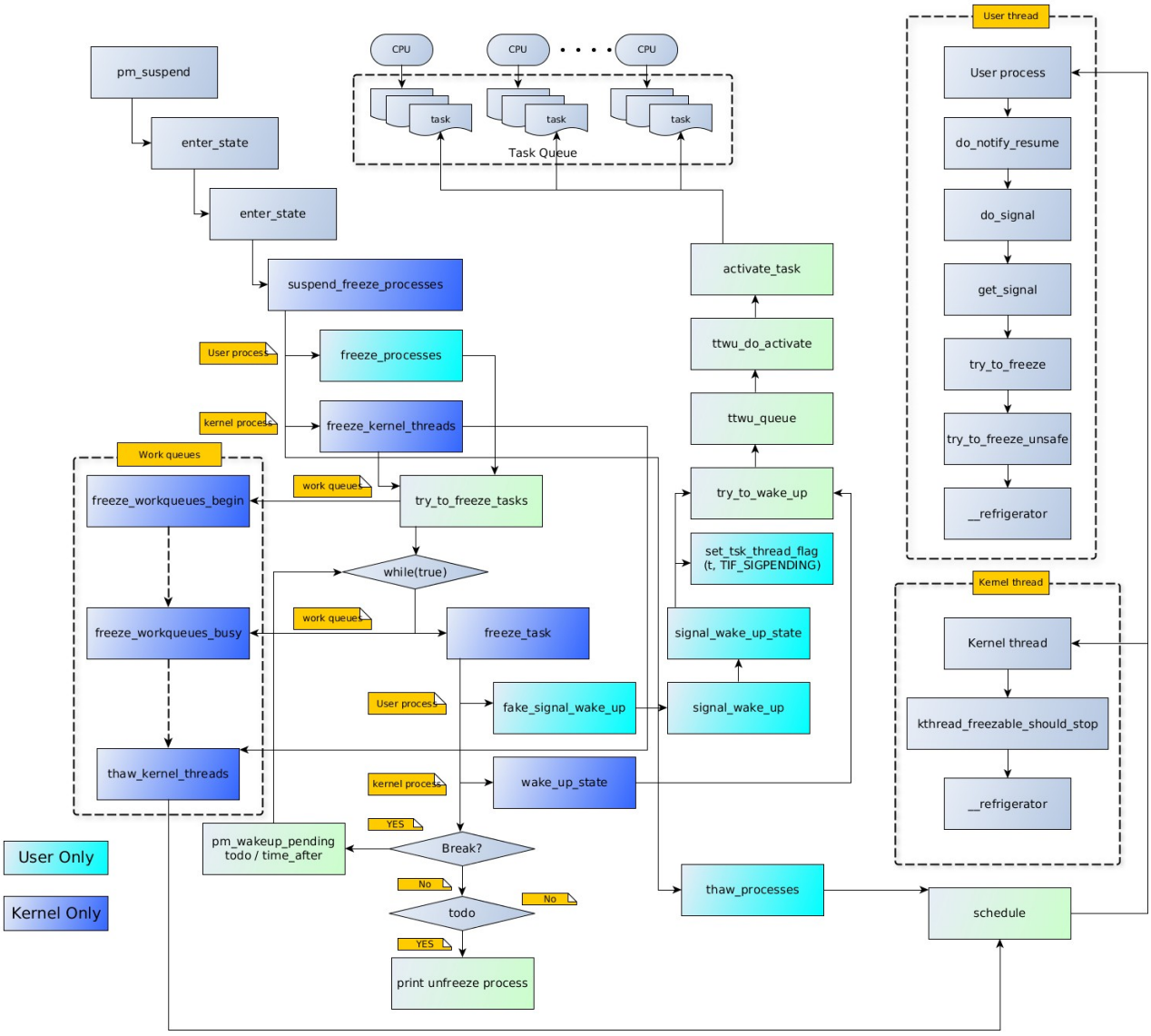
Sum up:



3.5 Freeze

3.5.1 Freeze process

Freeze process:



Key function

fake_signal_wake_up

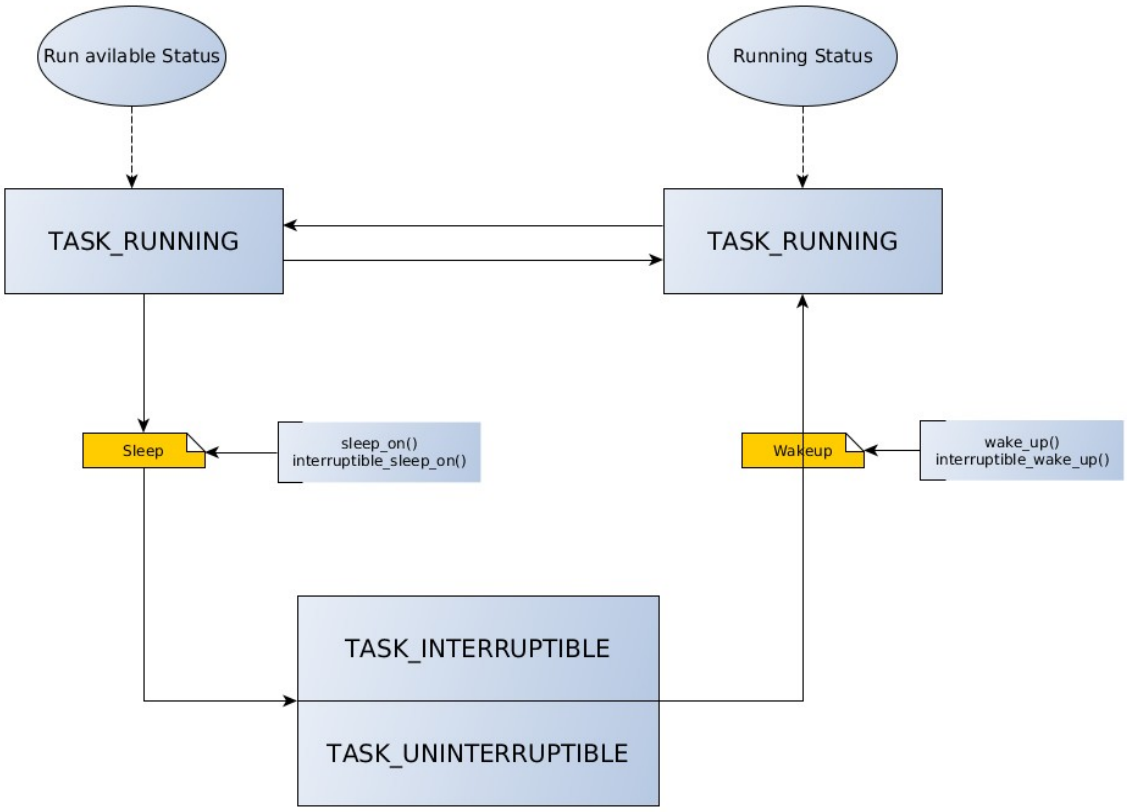
try_to_wake_up

set_tsk_thread_flag(t, TIF_SIGPENDING)

freeze_workqueues_begin

__refrigerator

Process status changed flow diagram:



3.6 CPU Idle

3.7 CPU HotPlug

3.8 Hibernate