Power Management Quality of Service (PM_QOS)

Mark.gross@intel.com
mgross@linux.intel.com

SSG Core Software Division

Introduction

- New HW provides more power/performance options than ever before.

  Device throttling over large dynamic ranges that can affect usability and device stability.

- Hardware devices talk in terms of latencies, time outs, and throughputs.
- Device drivers attempt to implement the power/performance policy in an information vacuum using only local (to the driver) knowledge.

Today's Situation

- Some PM architectures attempt to pull the policy implementation up to a centralized policy manager away from the drivers that know the hardware the best.
  - These create dual point maintenance of device power / performance knowledge -- a partial one in the driver and one in the policy manager.
  - Architecturally it removes all hope of good abstractions or stable and useful PM API's.

Enter PM QoS

- PM QoS provides a coordination mechanism between the hardware providing a power managed resource and users with performance needs
- It is a new kernel infrastructure to facilitate the communication of latency and throughput needs among devices, system, and users.
- Automatic power management, at the driver level, is enabled with coordinated device throttling given the QoS expectations on that device.

Talk outline

- Existing examples of PM-QoS
- Implementation walk through
- How to use it from user space
- How to use it from kernel space
- iwl4965 example of how you could put PM-QoS into your applications

Examples of pm_qos in the 2.6.25 Kernel

- CPU-IDLE provides processor C-states when idle. That is, it controls the idle processing and wakeup latency (dma-latency).
- ipw2100 malfunctions when C-state latencies are large
- pcm_native has sound artifacts when C-state latencies are large.

pm_qos_params.c

- implements a set of PM_QOS parameters (currently just cpu_dma_latency, network latency, network throughput), exported to the kernel and to user mode.
- maintains a list of pm_qos requests for each parameter, along with an aggregated performance requirement.
- maintains a kernel-only notification tree, for each parameter.
- provides the registration of performance requests and target change notification KAPIs.
- provides a user mode interface for requesting QoS, through simple character device file I/O.

Implementation

Maintains lists of requested performance values for each QOS parameter.

- when an element is added or changed, an aggregate target value is recomputed.
- if this aggregate target value changes, it invokes the notification tree for that QOS parameter.
- resources can also poll the aggregated value (see CPU_IDLE)

Interface

PM_QOS implements:

- A simple API for modifying the lists can be found in: pm_qos_params.h
- A user mode interface through simple character devices.

How to use PM_QOS from user mode:

- Register a default request value by opening one of the parameter device nodes.
- Update the request by writing a signed 32-bit integer to the open device node.
- Remove the request by closing the handle to the device node.
- Currently the device nodes are based on misc devices.

Python Example setting network_latency to at most 2000us

```
#!/usr/bin/python
import struct, time
DEV_NODE = "/dev/network_latency"
pmqos_dev = open(DEV_NODE,'w')
latency = 2000
data = struct.pack('=i', latency)
pmqos_dev.write(data)
pmqos_dev.flush()
while(1):
   time.sleep(1.0)
```

How a resource uses PM_QOS in the kernel

poll the aggregated target value.

KAPI:

- int pm_qos_requirement(int qos)

register a notifier into the parameter notification chain.

KAPIs:

- int pm_qos_add_notifier(int qos, struct notifier_block *notifier)
- int pm_qos_remove_notifier(...)

To create a new PMQOS parameter, you need to modify the pm_qos_init code.

How a dependent uses PM_QOS from kernel mode

- **register a named list element in the parameter list, along with an initial element target value.**
  - int pm_qos_add_requirement(int qos, char *name, s32 value)
- **update the value of the named element**
  - int pm_qos_update_requirement(int qos, char *name, s32 value)
- **clean up / remove named element**
  - void pm_qos_remove_requirement(int qos, char *name)

Aggregated target is recomputed after any change to a parameter list.

Notification trees are called if the aggregate value has changed.

Example application of PM_QOS in IWL4965 driver

- This is a work in progress.
- I'm currently working with one of the IWL 4965 developers to make this work.
- The 4965 has 6 high level power configurations effecting the powering of the antenna, how quickly it sleeps the radio and for how long between AP-beacons.
- Looks like a good application of PM_QOS network latency.

IWL4965 Main Power Configurations

- Power level zero radio is always on and powered.
- Power level five sleeps the radio as much as it can given the current access point beacon configuration.
- Power levels effect latency of incoming and outgoing packets and how quickly the device turns the radio back off when idle.
- There are a host of other unused power parameters in the driver that could be played with.

Network latency could be used to affect iwl4965 power policy

- today policy is set via sysfs and is specific to the 4965.
  - see store_power_level in iwl4965-base.c
- the iwl4965_init code could register for pm_qos notifications of updated network latency and execute a switch on the updated latencies to set new power levels upon pm_qos notification.
- with pm_qos use of network_latency other network devices could implement similar power / performance trade offs and enable sane user mode policy managers

use container_of to get a pdev from the notifier block call back

```
struct iwl4965_pm_qos_nb {
      struct notifier_block my_nb;
      struct pci_dev *pdev;
};


static int iwl4965_pm_qos_notify_call(struct notifier_block *nb,
   unsigned long val, void *v)
{
      struct iwl4965_pm_qos_nb *ipqn =container_of(nb,
           struct iwl4965_pm_qos_nb, my_nb);
     iwl4965_pm_qos_nb, my_nb);
     .....
```

User space can now set performance expectations on network latency

- network shooter games could set network latency to zero to disable all power management
- Web browser could set it for 2,000,000us
- IM application could set it for 500,000us
- User mode policy manager (OHM?) could set it to zero when on wall power and 10,000,000us when on battery.
- The above can happen at the same time, in any combination.

Do you have an application that could use PM-QoS?

Comments and Questions