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

Expanding the Vocabulary of Power Management

SSG Core Software Division

outline

- Vocab
- PM today and hope for a better tomorrow
- PM_QOS overview
- Range Timers overview
- Problematic mindsets
- PM categories
- discussion

Introduction

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

  Device throttling over bigger dynamic ranges that can effect usability and device stability.

- Hardware devices talk in terms of latencies, time outs, and throughputs.
- A lot of PM can be defined as an optimization problem to minimizing power use subject to performance and latency constraints
- A current fad in power management is to optimize for deep C-state latencies by system event consolidation

Vocabulary

- **Latency / Throughput**
- **PM-QoS**
  - Throttling Constraints
- **C-state residency**
  - Event consolidation
  - Range Timers
- **Platform throttling**
- **System throttling**
- **Process throttling**
- **User mode PM policy manager**
- **Implicit kernel power / performance policies**

Today's Situation

- Most device drivers attempt to implement the power/performance policy in an information vacuum and only local (to the driver) knowledge
- Typically device drivers implement a default policy of max performance unless told otherwise
- Embedded “applications” sometimes attempt to use modal power management, based on modal application stacks or root daemon processes.
  - At best using Sysfs interfaces that don't scale well across architectures and numbers of peripherals
- PM QoS provides a coordination mechanism.

tomorrow's vision

- **more drivers PM_QOS aware**
- **Process timer event consolidation and partial process throttling using range timers**
- **Platform throttling:**
  - Smarter hardware does automatic PM
    - sometimes needing hints from PM_QOS
  - IRQ consolidation
- **System throttling:**
  - More global timer event consolidation
  - Process throttling
  - More explicit power / performance default behaviors

Extending the notion of power management to throttling constraints:

- We want the OS to explicitly provide a best effort quality of service to the user programs
- Make Power management into a dynamic optimization problem minimizing the power utilization subject to usability constraints defined by user programs.
- Given good knowledge of the constraints the kernel can do a good job at minimizing the power used while running any workload.
- Provide for explicit power save default behaviors instead of implicit performance behaviors in the kernel.

pm_qos_params

- 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.
- Works only for parameters having a natural aggregation.
- maintains a kernel-only notification tree, for each parameter.
- provides the registration of performance requests and target change notification as KAPI's.
- provides a user mode interface for requesting QoS, through simple character device file I/O.

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.
- ipw2100 depends on C-state latencies
- The dependents on the C-State latencies are pcm_native and ipw2100, which have problems with longer wake up latencies.

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)
```

User space can now set performance expectations on network latency

- network game 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 could set it to small-num when on wall power and 10,000,000us when on battery.
- The cool thing is all of the above can happen at the same time, in any combination, and the 4965 will do right thing(tm)

Range timers

- **Is a new way of throttling processes.**
- **Effects select and process timers.**
- **Created to solve the “Flash is melting my system” problem.**
- **Defines a latency value for wake up's (“slop”)**
- **Timer code consolidates wake ups of “sloppy” events with other events.**
- **Implementation is process based and children processes inherit slop from parent**
- **PCTL ABI for user mode to change slop value of a given PID.**
  - Expect “nice” program / api for changing behavior.
- **Its a step in the right direction.**

Range Timers comments

- Defaults are heuristics
- Its a solution to a specific problem and may not generalize well.
- Doesn't allow for global system adjusting
- Its design will lead to the existence of daemon to watch for piggy processes, with blacklist, for throttling things when they are burning too much CPU

Problematic PM design mindsets

- PM architectures that attempt to pull the policy implementation up to a centralized policy manager away from the drivers that know the hardware the best.
  - Creating 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 for more stable and useful PM API's.
  - Scales poorly across architectures and devices

Categories of PM methods

- **Automatic**
  - Demand based
  - Steepest descent constraint based
- **Platform**
  - Smart hardware needing QOS hints
  - S-States
- **System**
  - Event consolidation
  - On Demand
- **Process**
  - Range timers
- **Modal**
  - Global system PM control

Not everything is a nail

- User mode root daemon knows all and forcibly controls all, but is not portable
- Kernel knows best on its own, but needs some policy input
- One parameter applies globally, except for special cases
- PM_QOS is good for best effort, throttling constraining, but not for forced throttling or suspend

Discussion