Power Management Quality-of-Service

Linux Power Management

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

What is “power management QOS”?

Quality-of-service:

- Defines minimum level of service required
- CPU interrupt latency or throughput
- Network latency
- ...

What is “power management QOS”?

“What’s the ‘power management’ part, then?”

- Guilt-by-association, mostly :-)

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

Advisory Framework

It’s mostly notifiers:

- Only platform subsystems can truly implement
- Very generic, by necessity

Three basic parameter classes:

- Latency
- Timeout (*)
- Throughput

Advisory Framework

```
struct notifier_block nb;
int my_callback(struct notifier_block *nb,
                unsigned long event,
                void *ignored)
{
  ...
}
```

Advisory Framework

```
/* add our notifier callback to the chain */
nb.notifier_call = my_callback;
notifier_chain_register(&chain, &nb);

/* elsewhere, invoke the chain of callbacks */
/* (i.e. when the event associated with the chain occurs) */
notifier_call_chain(&chain, event, data);
```

Advisory Framework

Existing service classes:

- `PM_QOS_CPU_DMA_LATENCY`
- `PM_QOS_NETWORK_LATENCY`
- `PM_QOS_NETWORK_THROUGHPUT`

(It’s fairly easy to create more as needed)

struct pm_qos_request

```
struct pm_qos_request {
  struct plist_node node;
  int pm_qos_class;
  struct delayed_work work;
};
```

`pm_qos_add_request()`

Expresses an initial quality level:

- Use only once, e.g. during `probe()`
- Used internally to “hook in” the associated notification

pm_qos_add_request()

```
struct foo {
  ...
  struct pm_qos_request qos;
  ...
};
```

pm_qos_add_request()

```
int foo_probe(...)
{
  ...
  /* usecs */
  latency = PM_QOS_CPU_DMA_LAT_DEFAULT_VALUE;
  pm_qos_add_request(&foo->qos,
      PM_QOS_CPU_DMA_LATENCY, latency);
  ...
}
```

pm_qos_update_request()

Expresses a change in quality requirement:

- E.g. your device is at a different activity level
- Triggers a notification callback, as with the initial request

pm_qos_update_request()

```
int foo_open(...)
{
  ...
  pm_qos_update_request(&foo->qos, 100 /* usecs */);
  ...
}

int foo_close(...)
{
  ...
  pm_qos_update_request(&foo->qos, default_latency);
  ...
}
```

pm_qos_update_target()

Callback to install a new quality target:

- For constraint implementers, not users
- Not normally invoked directly

pm_qos_update_target()

```
int pm_qos_update_target(...)
{
  ...
  blocking_notifier_call_chain(...);
  ...
}
```

`pm_qos_add_notifier()`

Installes a notifier:

- Bridges gap between requestors, implementers
- Use this when your device can *assert* quality-of-service

drivers/cpuidle/cpuidle.c

```
int cpuidle_latency_notify(struct notifier_block *b,
                unsigned long l, void *v)
{
  smp_call_function(smp_callback, NULL, 1);
  return NOTIFY_OK;
}

struct notifier_block cpuidle_latency_notifier = {
  .notifier_call = cpuidle_latency_notify,
};

void latency_notifier_init(struct notifier_block *n)
{
  pm_qos_add_notifier(PM_QOS_CPU_DMA_LATENCY, n);
}
```