Power saving in Linux devices

Chris Simmonds
2net Limited

Class 5.5
Embedded Systems Conference UK. 2009

Overview

- Power saving is important because
  - Extends battery life
  - Conforms to regulations for consumer and office equipment
- Micro power management
  - Reduce clock speeds, sleep during idle - little impact on overall performance
- Macro power management
  - Suspend and hibernate

Micro power management

- Techniques that have low latency
  - i.e. Low overhead in time and power switching from one state to another
- CPUFreq
  - Scale core clock frequency dependant on load
- Dynatick (Tickless operation)
  - Fewer timer interrupts, more time sleeping
- CPUIdle
  - Deeper sleep when idle

CPUFreq

- Requires chip support to set core frequency
- Need to know latency (overhead) of changing frequency
- Policy set by “governor”
- Typically:
  - Increase frequency as processor load increases
  - Reduce frequency when load drops

CPUFreq governors

powersave - always select the lowest frequency

performance - always select the highest frequency

ondemand - change frequency based on utilisation: if the CPU is idle < 20% of the time set the frequency to the maximum; if idle >=30% drop the frequency down in 5% decrements

conservative - as “ondemand”, but switches to higher frequencies in 5% steps rather than going immediately to the maximum

userspace - frequency is set by a userspace application

CPUFreq settings

- In /sys/devices/system/cpu/cpu0/cpufreq/
  - scaling_max_freq and scaling_min_freq and scaling_available_frequencies
  - scaling_available_governors which lists the names of the built-in governors
  - scaling_governor which tells you the current governor and allows you to set a new one
  - scaling_setspeed allows you to set the speed if the governor is set to “userspace”

User space governors

- The “userspace” governor allows the frequency to be set by writing to scaling_setspeed
- Examples
  - cpuspeed [2], cpudyn [3] and cpufreqd [4]

Dynamic tick

- Historically Linux has a regular timer tick
  - HZ from 100 to 1000
- Can create unnecessary wakeups
- Dynamic tick mode means timer interrupts occur only when needed
  - CONFIG_NO_HZ in Linux 2.6.21

Dynamic tick: before

```
# echo 1 > /proc/timer_stats; sleep 10; echo 0 > /proc/timer_stats
# cat /proc/timer_stats
Timer Stats Version: v0.2
Sample period: 10.079 s
 1008,     6 khelper          tick_setup_sched_timer (tick_sched_timer)
   11,     1 swapper          phy_start_machine (phy_timer)
   40,     1 swapper          usb_hcd_poll_rh_status (rh_timer_func)
    5,     1 swapper          schedule_delayed_work_on (delayed_work_ti
    2,     0 swapper          page_writeback_init (wb_timer_fn)
    2,   275 thttpd           schedule_timeout (process_timeout)
    2,     5 events/0         __netdev_watchdog_up (dev_watchdog)
    2,     0 swapper          neigh_add_timer (neigh_timer_handler)
    1,     1 swapper          neigh_table_init_no_netlink (neigh_period
    1,   709 sleep            do_nanosleep (hrtimer_wakeup)
1074 total events, 106.558 events/sec
```

Dynamic tick: after

```
# echo 1 > /proc/timer_stats; sleep 10; echo 0 > /proc/timer_stats
# cat /proc/timer_stats
Timer Stats Version: v0.2
Sample period: 10.061 s
    8,     0 swapper          tick_nohz_restart_sched_tick (tick_sched_t
   40,     1 swapper          usb_hcd_poll_rh_status (rh_timer_func)
   54,     0 swapper          tick_nohz_stop_sched_tick (tick_sched_time
    5,     1 swapper          schedule_delayed_work_on (delayed_work_tim
   10,     1 swapper          phy_start_machine (phy_timer)
    2,   275 thttpd           schedule_timeout (process_timeout)
    2,     5 events/0         __netdev_watchdog_up (dev_watchdog)
    2,     0 swapper          page_writeback_init (wb_timer_fn)
    1,     5 events/0         queue_delayed_work (delayed_work_timer_fn)
    1,     1 swapper          neigh_table_init_no_netlink (neigh_periodi
    1,   281 sleep            do_nanosleep (hrtimer_wakeup)
126 total events, 12.523 events/sec
```

CPUIdle

- Normally idle task == halt
- CPUIdle allows for deeper sleep modes
  - stopping the clock to some parts of the core
  - powering down parts of the core, losing some state

ARM 11 power modes

Mode	ARM core	RAM arrays	Wake-up mechanism
Run Mode	Powered-up: everything clocked	Powered-up	N/A
Standby	Powered-up: only wake-up logic clocked	Powered-up	Wake-up on interrupts (external or timer/ WD).
Dormant	Powered-off	Retention state/voltage	External wake-up event to power controller.
Powered- off	Powered-off	Powered-off	External wake-up event to power controller.

The CPUIdle driver

- Call `cpuidle_register_device()` to register callbacks and parameters, including
  - Number of power saving states
  - Power consumption (mW)
  - Exit latency (micro seconds)
- Only one driver in the kernel source
  - For PC with ACPI
  - See `drivers/acpi/processor_idle.c`

Example CPUIdle states

Taken from a laptop with Intel CPU and ACPI states C0 .. C3

State	Power (mW)	Latency (uS)
C0	-1	0
C1	1000	1
C2	500	1
C3	100	57

CPUIdle Governors

ladder - steps down or up sleep states one at a time depending on the time spent in the last idle idle period. It works well with a regular timer tick, but not with dynamic tick

menu - selects sleep state based on expected idle time. Works well with dynamic tick systems.

CPUIdle user interface

- In `/sys/devices/system/cpu/cpuidle`
  - *current_driver* - the name of the cpuidle driver
  - *current_governor_ro* - the name of the governor
- In `/sys/devices/system/cpu/cpu0/cpuidle`
  - desc : description of the idle state
  - latency : latency of this idle state (microseconds)
  - name : name of the idle state (string)
  - power : power consumed while in this idle state (in milliwatts)
  - time : total time spent in this idle state (in microseconds)
  - usage : Number of times this state was entered (count)

Power QOS

- Power management can impact some work loads
- Power management Quality Of Service added in 2.6.25
- Defines minimum latency for CPU and network
- Applications can write desired latency (uS) to
  - /dev/cpu_dma_latency
  - /dev/network_latency

Macro power management

- Put system into a suspend mode:
  - full power
  - reduced power
  - suspend
  - hibernate

reduced power, suspend } suspend-to-ram

hibernate } suspend-to-disk

Suspend to RAM

- Freeze all tasks
- Suspend all devices
- (Usually) put the DRAM into a self-refresh mode
- Set the CPU into the deepest sleep state and wait for a wake-up event
- On wake-up, set the DRAM to normal refresh mode
- Resume all devices
- Thaw all tasks

Suspend-to-disk

- Freeze all tasks
- Suspend all devices
- Take a snapshot of the system image and store it in a swap partition on disk
- Power off
- On boot up, kernel tests for a valid image and loads if found
- Suspend then resume all devices to bring them into the same state as when the image was created
- Thaw all tasks

Driver support for suspend & resume

- Drivers that want to participate must implement
  - suspend - set device into power saving state (see next slide)
  - resume - return device to normal operation

Suspend states

- PM_EVENT_SUSPEND - stop all activity and put the device into a low power state
- PM_EVENT_HIBERNATE - as above, put enable wake-up events
- PM_EVENT_FREEZE - stop all activity but don't switch to a low power mode
- PM_EVENT_PRETHAW - a warning that a “suspend-to-disk” image is about to be loaded: set the hardware into a compatible state. Drivers that implement this often simply reset the device.

Wakeup events

- Driver that can wake from suspend mode should set `can_wakeup` flag
  - Examples: buttons, keyboard, touch screen, RTC
- Wakeup events can be selectively disabled (next slide)
  - Driver should call `device_may_wakeup()` in suspend to see if hardware should be armed for wakeup

Power management user interface

- Possible states are in /sys/power/state
  - suspend - lightweight suspend-to-ram
  - mem - full suspend-to-ram
  - disk - suspend-to-disk
- Set state by writing the state to the file, e.g.

```
echo “mem” > /sys/power/state
```

Summary

- Micro power management
  - CPUFreq, Dynamic tick, CPUIdle
- Macro power management
  - Suspend (to RAM) and hibernate (to disk)
  - Hibernate requires fast mass storage - flash memory too slow
- Good power management requires support in
  - Board support package
  - Drivers
  - applications

References

[1] IEA “1 Watt plan”
http://www.iea.org/

[2] cpuspeed
http://www.carlthompson.net/Software/CPUSpeed

[3] cpudyn
http://mnm.uib.es/gallir/cpudyn/

[4] cpufreqd
http://www.linux.it/~malattia/wiki/index.php/Cpufreqd

[5] Driver core API changes for 2.6.19
http://lwn.net/Articles/201111/

[6] Clockevents and dyntick in 2.6.21
http://lwn.net/Articles/223185/