Energy Efficiency in Operating Systems

Björn Brömstrup

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

December 3, 2014

Outline

ACPI

Standardized interface for power management

- Global states: G0 – G3
- Suspend states: S1 – S4
- Device states: D0 – D3
- CPU idle states: C0 – Cn
- CPU performance states: P0 – Pn

Devices

- D0 — on, D3 — off
- D1 and D2 are not necessarily available
- Most power management happens either in the specific device driver or in userspace
- Power domain hierarchy

  Some devices might depend on others for power
- Operating system can automatically suspend devices without held references

CPU

In a typical system the CPU is the biggest power-draw (apart from the GPU, depending on workload)

Strategies

- During idle:
    - Removing timer interrupts (sleeping longer)
    - Choosing the right idle state
- Under load:
    - CPU frequency scaling
    - Load balance over multiple CPUs

Scheduler and timer

- The **scheduler** allocates CPU time to individual processes
- Via **interrupts**, the CPU is literally interrupted in its current execution to deal with new workloads
- Programmable **timer interrupts** keep track of future workload

- Baring any hardware interrupts, the CPU has a good idea how much work happens in the near future
  - → We roughly know how much we can idle
- Likelihood of hardware interrupts can be estimated, based on runtime statistics

Ticks and timers

Traditional system

- Periodic tick: Scheduler runs in a constant interval (on Linux: 100Hz – 1000Hz)
  $\rightarrow$ constant wakeups
- No concerns for energy efficiency

Now

- Dynamic tick: Program the next timer interrupt to happen only when work needs to be done
- Deferrable timers: Bundle unimportant timer events with the next interrupt
- Timer migration: Move timer events away from idle CPUs

CPU idling

- Entering/exiting deeper idle states takes more time
  - → Trade-off between idle state and CPU latency
- Switching idle state takes energy
  - → Idling for too little time can *cost* energy
- Deeper idle states will switch off more and more parts of the CPU
  - → Invalidation of cache contents and the subsequently necessary restore can mean additional performance impacts

cpuidle

cpuidle is a Linux kernel subsystem to manage CPU idling

- The decision of which idle state to choose is delegated to one of two governors
  - ladder
  - menu
- Governors rate themselves on how effective they are on a given system and the one with the higher rating is chosen
- Constraints, like latency requirements, are tracked with a Quality of Service (QoS) subsystem

cpuidle

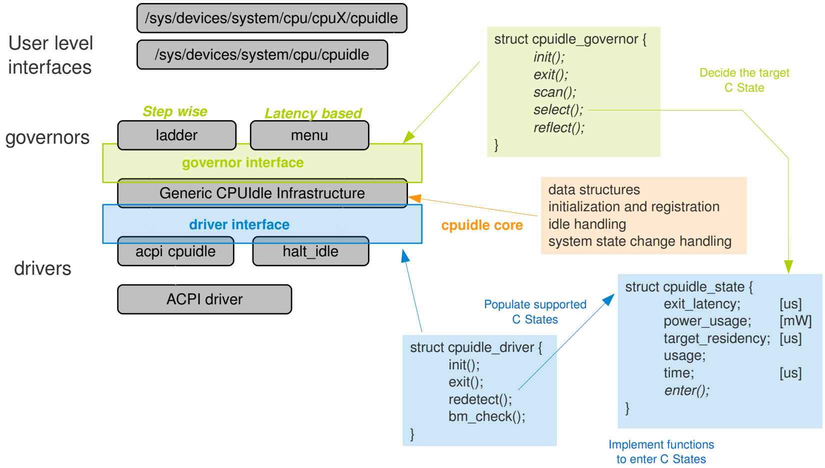

Figure: cpuidle in the Linux kernel

source: Patrick Bellasi, Linux Power Management Architecture, http://ilinuxkernel.com/Backup/Data/Linux.Power.Management.Architecture.-.A.review.on.Linux.PM.frameworks.December.2010.pdf

Ladder governor

Ladder governor

- Simple, step-based approach
- Works well with periodic tick

```
if (latency requirements aren't fulfilled)
    jump to higher state
else if (last idle time > up threshold)
    sleep deeper
else if (last idle time < down threshold)
    sleep lighter
```

Menu governor

- Tries to select the optimal state
- Looks at a variety of constraints
  - Latency requirements
  - Energy break even point

    Transitioning idle states costs energy
    $\rightarrow$ Not idling long enough is wasteful
  - Performance impact

    The busier the system, the more conservative our choice of idle state
  - Expected sleep time

    When is the next timer interrupt and what is the likelihood of hardware interrupts?

Dynamic Voltage and Frequency Scaling (DVFS)

To reduce energy consumption CPU performance can be reduced

- Race to idle vs. working longer at lower frequency
- Rapid frequency switching made it possible to adjust the frequency dynamically based on workload
- Frequency itself is not a big power draw, but to reduce CPU voltage, the frequency has to be reduced first
- Power consumption scales quadratically with CPU voltage
- There may be power dependencies between CPUs on the same socket

cpufreq

cpufreq is a Linux kernel subsystem to manage CPU frequency states and changes

- A policy is a frequency range in which the CPU needs to stay

  The policy is determined through hardware constraints and explicit setting in userspace

- Governors decide which P-state within the current policy to choose
- The active governor decides by itself when to switch frequency. It is not called by the scheduler

cpufreq

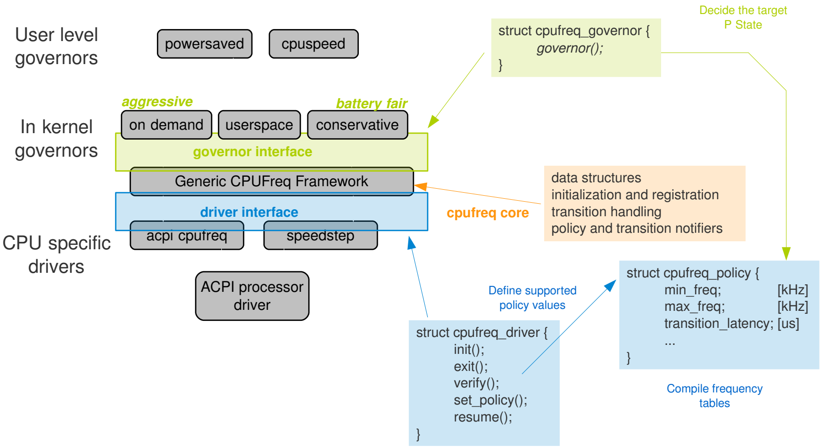

Figure: cpufreq in the Linux kernel

source: Patrick Bellasi, Linux Power Management Architecture, http://ilinuxkernel.com/Backup/Data/Linux.Power.Management.Architecture.-.A.review.on.Linux.PM.frameworks.December.2010.pdf

Simple governors

- performance
  - Keeps the CPU at the highest frequency
- powersave
  - Keeps the CPU at the lowest frequency
- userspace
  - Let's userspace set the frequency
  - Programs: powersaved, cpuspeed
  - Larger overhead

Ondemand governor

drivers/cpufreq/cpufreq_ondemand.c:

> Every sampling_rate, we check, if current idle time is less than 20%, then we try to increase frequency. Else we adjust the frequency proportional to load.

- Every frequency increase jumps to 100%
  - Minimizes performance impact
  - Utilizes race-to-idle
- Sysfs parameters
  - sampling_rate
  - up_threshold
  - ignore_nice_load
  - sampling_down_factor
  - powersave_bias

Conservative governor

- Less aggressive frequency scaling
- Is a little more energy-efficient under light load

```
for every CPU
  every X milliseconds
    if (utilization since last check > 80%)
      increase frequency by 5%
  every Y milliseconds
    if (utilization since last check < 20%)
      decrease frequency by 5%
```

Future direction: The energy-aware scheduler

- Right now, the scheduler is optimized to get work done as quickly as possible
- In a multicore environment, that means processes are spread out among CPUs with no consideration to energy-cost
- Idea 1: Consolidate processes on fewer power domains, whenever possible
- Idea 2: Bundle workloads to as few CPUs as possible without sacrificing performance

Future direction: The energy-aware scheduler

Problems

- Discrimination between "small tasks" and "big tasks"
- Finding the right distribution between CPUs is difficult and can be costly
- Interaction between scheduler, cpuidle and cpufreq is complicated and suboptimal
- Further complication: non-homogeneous CPU architectures, e.g. ARM big.LITTLE

Conclusion

- Modern systems are very efficient at doing nothing or doing a lot
- Energy-efficiency under medium load is complicated

Thank you for listening.
Any questions?

Kernel sources and documentation

- Documentation/cpuidle/*
- Documentation/cpu-freq/*
- Documentation/scheduler/*
- Documentation/timers/*
- drivers/cpufreq/cpufreq*
- drivers/cpuidle/cpuidle*
- include/linux/cpufreq.h
- include/linux/cpuidle.h
- kernel/sched/idle.c

References

- Patrick Bellasi, Linux Power Management Architecture, http://ilinuxkernel.com/Backup/Data/Linux.Power.Management.Architecture.-.A.review.on.Linux.PM.frameworks.December.2010.pdf
- Vaidyanathan Srinivasan, Gautham R Shenoy, Srivatsa Vaddagiri, Dipankar Sarma, Energy-aware task and interrupt management in Linux, https://www.kernel.org/doc/ols/2008/ols2008v2-pages-187-198.pdf
- Venkatesh Pallipadi, Shaohua Li, Adam Belay, cpuidle — Do nothing, efficiently..., https://www.kernel.org/doc/ols/2007/ols2007v2-pages-119-126.pdf
- Venkatesh Pallipadi, Alexey Starikovskiy, The Ondemand Governor, https://www.kernel.org/doc/ols/2006/ols2006v2-pages-223-238.pdf
- Len Brown, Anil Keshavamurthy, David Shaohua Li, Robert Moore, Venkatesh Pallipadi, Luming Yu, ACPI in Linux, https://www.kernel.org/doc/ols/2005/ols2005v1-pages-59-76.pdf
- Rafael J. Wysocki, Runtime Power Management Framework, https://events.linuxfoundation.org/slides/2011/linuxcon-japan/lcj2011_wysocki2.pdf
- Rafael J. Wysocki, Power Management In The Linux* Kernel, http://events.linuxfoundation.org/sites/events/files/slides/kernel_PM_plain.pdf
- Tate Hornbeck, Peter Hokanson ,Power Management in the Linux Kernel ,www.ruf.rice.edu/ mobile/elec518/lectures/2011-tatepeter.pdf
- corbet, Deferrable timers, http://lwn.net/Articles/228143/
- corbet, Clockevents and dyntick, http://lwn.net/Articles/223185/
- Jonathan Corbet, Per-entity load tracking, http://lwn.net/Articles/531853/
- Jonathan Corbet, Power-aware scheduling meets a line in the sand, http://lwn.net/Articles/552885/
- Libby Clark, Boosting Linux Power Efficiency with Kernel Scheduler Updates, https://www.linux.com/news/featured-blogs/200-libby-clark/715486-boosting-linux-power-efficiency-with-kernel-scheduler-updates/
- Preeti U. Murthy, Overview of the Current Approaches to Enhance the Linux Scheduler, https://events.linuxfoundation.org/images/stories/slides/lfcs2013_murthy.pdf