Politecnico di Milano

Power Management at OS level

saving power with Linux

Lecturer:
Ing. Patrick Bellasi
Politecnico di Milano - DEI
bellasi@elet.polimi.it

Outline

Power Savings Basics

CPU Power Management

- CPU Idling
- Tickless idling
- CPU Frequency Scaling
- CPU Throttling

Device and Bus power management

- backlights
- hard disks
- bus Devices (PCI, USB, PCMCIA et al.)
- device driver tuning

Applications power management

Power Savings Basics

Uninterrupted sleep

- how energetic would you feel in the morning if someone had been waking you up every five minutes during the night?

Recharging without interruptions

- power management hasn't been a conscious focus of software design up to now
  - it's not surprising that computers are far less energy efficient than they could be
  - applications should wake-up the system when they need something, so the system has an opportunity to recharge in the downtime
- the less it's interrupted, the more and better recharging can be done

Race to idle

- run hard and then sleep hard and long
  - his method only works if you save a lot of energy while you're sleeping. If you can't, then it's better to work slower.

Simple Savings

Turn it off

- if you don't use a device, turn it off
  - Even if you're not using it, if it's on, it's still consuming power
  - e.g. if you're not playing audio, close your audio player application

Create long idle time between activities

- processors (but also disks and many other components) have several degrees of saving power when they are idle
- if a component is idle for a long time, it can go into the deepest possible power saving state, while if it's only idle for a short time, it can only go into a shallow power saving state
- it depends on the component type what defines "long time"
  - for processors, a "long time" is roughly around 20 milliseconds, while for disks this is around 2 seconds

it is better to do all the work in one step than to have several smaller steps

Outline

Power Savings Basics

CPU Power Management

- CPU Idling
- Tickless idling
- CPU Frequency Scaling
- CPU Throttling

Device and Bus power management

- backlights
- hard disks
- bus Devices (PCI, USB, PCMCIA et al.)
- device driver tuning

Applications power management

CPU Power Management

CPUs consume large amounts of energy

Several different techniques attempt to reduce this

If there is no work to do:

- ***Idling:*** the processor is put into an low-power idle state
- ***Frequency scaling:*** the frequency the CPU operates at can be modulated
- ***Throttling:*** the CPU can be forced to a non-working state for short periods of time

CPU Idling

Possibly the most important runtime power management technique

- under normal operation the CPU only needs to execute code once in a while
- e.g. on the author's system the CPU is only needed approx 4% of the time while writing this slides

When there is no work to do:

- certain parts of the CPU can be shut down and re-activated once they are required for operation again
- On modern processors there exist multiple different such “idle states” **[acpi]**
- treadoff between latency and power saving opportunities

CPU Idling - CPU Power States

A microprocessor, when it's not executing instructions, can save power in several ways (called "states"):

- **C-states** - a set of idle states
- **P-states** - performances states, which allow you to scale the frequency in voltage of your CPU
- **T-states** - thermal states that allow the system to respond to emergency thermal conditions

Each of those ways has a different *tradeoff* in terms of power saving versus latency and performance

C-state	Max Power Consumption
C0 (busy wait)	35 Watts (TDP)
C1	13.5 Watts
C2	12.9 Watts
C3	7.7 Watts
C4	1.2 Watts

source:
Intel® Core™2 Duo datasheet

The Power-Performance Tradeoff

The deeper the C-state, the longer it takes to leave the C-state, and the more energy this transition costs

- if your system is going in and out of idle at a high frequency, say every millisecond, using deeper states during the short idle periods could consume more energy than that conserved during its very brief resting period
- if your system is actually idle for longer periods of time, say 20ms, then deeper states becomes the clear winner

  ...unless the exit latency cannot be tolerated

Need to better exploit deeper power saving states

- the system should wake up only when there actually is a new task to run or interrupt activity
- with the Linux kernel relying heavily on the concept of "jiffies" for timers and fair scheduling this is not possible

the solution is a tickless kernel

The Kernel's Timer Tick

Until 2.6.21, the Linux kernel programmed the PIT chip of the PC to generate interrupts at a regular interval of either 250 Hz or 1000 Hz

- timer tick approach has a certain elegance in it's simplicity and has served Linux well since the early 1990's
  - increment the "jiffies" variable, deferred events (timers), process accounting, process scheduler time slicing
- regular 1ms or 4ms interrupt has the effect of waking the CPU frequently from the deep sleep states, or even preventing the CPU from ever entering the deepest sleep states, which obviously makes the system consume more power than needed
- only first two functions are really relevant when the cpu is idle

Clock Devices Classification

Clock Sources

- devices that can answer the "what time is it right now" question
- clock source infrastructure abstracts away the differences between the various devices towards the rest of the kernel

Instead of updating jiffies by one every timer tick, jiffies gets updated to what the value should be based on the "what time is it" question to the clock source layer

Event Sources

- devices that can generate an interrupt after a software-specified amount of time
- event source layer abstracts these various devices for the rest of the kernel, and picks the best one for a task based on the various capabilities, e.g. in terms of precision, accuracy and maximum duration

Instead of looking every millisecond if any timer is due for processing, the kernel calculates when the first timer is due and asks the event source layer to give a single interrupt at exactly the right time

Tickless Idle

the removal of the regular timer tick when idle

- introduced by Thomas Gleixner et al, become part of the 2.6.21 kernel for the i386 architecture

potential for power saving

- in principle: by not having a regular timer tick when the processor is idle, it is possible to have really long periods of idle as long as there are no future timers planned
- in practice: unfortunately, on a current Linux distribution, both the kernel and userspace applications set so many timers that it is not uncommon to have 500 or more of such events per second...

both kernel and userspace side needs fixing

CPU Idling: needed fixes

To better support CPU idling some fixes are needed:

- kernel side
  - some driver's "randomly short" timers can be increased without any noticeable effects to the system or the user
  - align as many timers as possible to happen at the same time
  - new kernel API: `round_jiffies`
- userspace side
  - many desktop programs behave really badly and have a large number of timers that aren't really needed
    - the most frequent scenario is code polling for something while it would just get an event if the programmer had bothered
  - programs that poll frequently are actively hurting the power consumption of a tickless kernel
  - new glib API providing rounding and grouping timers: `g_timeout_add_seconds`

PowerTOP

A Linux tool that helps you find those programs that are misbehaving while your computer is idle

Goals:

- show how well your system is using the various hardware power-saving features
- show you the culprit software components that are preventing optimal usage of your hardware power savings
- help Linux developers test their application and achieve optimal behavior
- provide you with tuning suggestions to achieve low power consumption

PowerTOP

```
File  Edit  View  Terminal  Go  Help
     PowerTOP version 1.8       (C) 2007 Intel Corporation

Cn                  Avg residency       P-states (frequencies)
C0 (cpu running)          (12.9%)         1.71 Ghz     9.8%
C1                  0.0ms ( 0.0%)         1200 Mhz     0.3%
C2                 10.7ms (87.1%)          800 Mhz     0.5%
C3                  0.0ms ( 0.0%)          600 Mhz    89.4%
C4                  0.0ms ( 0.0%)

Wakeups-from-idle per second : 81.2     interval: 15.0s
Power usage (ACPI estimate): 14.1W (6.6 hours) (long term: 136.4W,/0.7h)

Top causes for wakeups:
  34.4% ( 31.9)       <interrupt> : ipw2200, Intel 82801DB-ICH4, Intel 82801DB-I
  19.4% ( 18.0)       firefox-bin : futex_wait (hrtimer_wakeup)
  15.5% ( 14.4)                 X : do_setitimer (it_real_fn)
  11.5% ( 10.7)         evolution : schedule_timeout (process_timeout)
   4.3% (  4.0)   <kernel module> : usb_hcd_poll_rh_status (rh_timer_func)
   3.9% (  3.6)       <interrupt> : libata
   1.8% (  1.7)     <kernel core> : sk_reset_timer (tcp_delack_timer)
   1.2% (  1.1)                 X : schedule_timeout (process_timeout)
   1.1% (  1.0)          Terminal : schedule_timeout (process_timeout)
   1.1% (  1.0)       xfce4-panel : schedule_timeout (process_timeout)
   0.6% (  0.5)   <kernel module> : neigh_table_init_no_netlink (neigh_periodic_
   0.5% (  0.5)             spamd : schedule_timeout (process_timeout)
   0.5% (  0.5)          events/0 : ipw_gather_stats (delayed_work_timer_fn)
   0.4% (  0.3)         xfdesktop : schedule_timeout (process_timeout)
   0.4% (  0.3)       firefox-bin : sk_reset_timer (tcp_write_timer)
   0.3% (  0.3)             nscd : futex_wait (hrtimer_wakeup)
   0.2% (  0.2)       xscreensaver : schedule_timeout (process_timeout)
   0.2% (  0.2)         ksnapshot : schedule_timeout (process_timeout)

Suggestion: Disable the unused bluetooth interface with the following command:
  hciconfig hci0 down ; rmmod hci_usb
Bluetooth is a radio and consumes quite some power, and keeps USB busy as well.
 Q - Quit   R - Refresh   B - Turn Bluetooth off
```


CPU Frequency Scaling

Perhaps the best-known runtime power management technique

Intel(R) SpeedStep Technology, AMD PowerNow! and Cool&Quiet!, Transmeta Longrun, ...

If the CPU clock frequency is lowered:

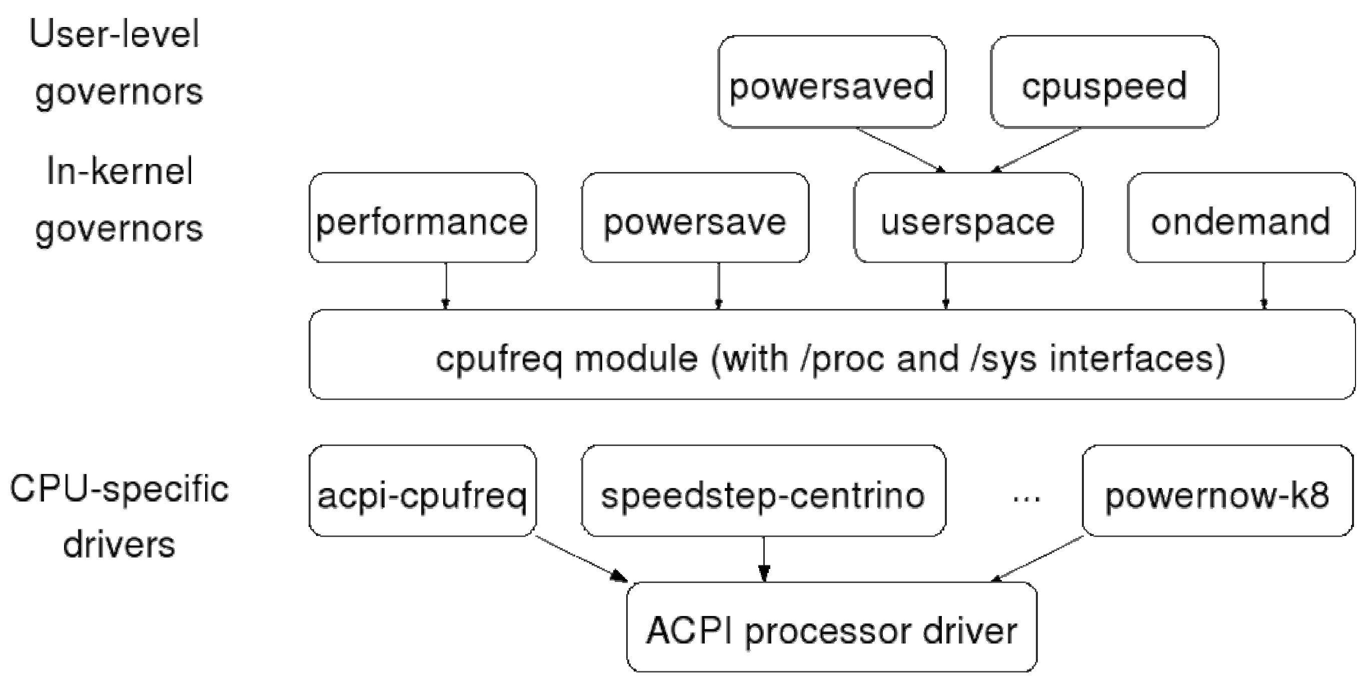
- the energy consumption is reduced linearly
- the voltage driving the CPU can be lowered as well
- highly increased **"instruction per energy consumption"** ratio

if you accept to wait longer for the result,
you can compute much more data with the same amount of energy

In the Linux kernel, support for CPU frequency and voltage scaling is provided by the cpufreq subsystem in all 2.6. kernels

The CPUFreq Subsystem

Provides a modularized set of interfaces to manage the CPU frequency changes



The CPUFreq Subsystem (cont)

Cpufreq module

- provides a common inter face to the various low-level, CPU-specific frequency control technologies and high-level CPU frequency controlling policies
- decouples the CPU frequency controlling mechanisms and policies and helps in independent development of the two
- provides some standard interfaces to the user, with which the user can choose the policy governor and set parameters for that particular policy governor

The CPUFreq Subsystem (cont)

CPU-specific drivers

- implement different CPU frequency changing technologies
  such as Intel R SpeedStep R Technology, Enhanced Intel SpeedStep Technology, AMD PowerNow! , and Intel Pentium 4 processor clock modulation
- on a given platform, one or more frequency modulation technologies can be supported, and a proper driver must be loaded for the platform to perform efficient frequency changes
- the cpufreq infrastructure allows use of one CPU-specific driver per platform
- some of these low-level drivers also depend on ACPI methods to get information from the BIOS about the CPU and frequencies it can support

The CPUFreq Subsystem (cont)

In-kernel governors

- the cpufreq infrastructure allows for frequency-changing policy governors, which can change the CPU frequency based on different criteria such as CPU usage
- the cpufreq infrastructure can show available governors on the system and allows the user to select a governor to manage the frequency of each independent CPU
- Kernel 2.6.16 comes bundled with five different governors:
  - *performance, powersave, userspace, ondemand and conservative*
  - the first three of these governors can be run on any kind of CPU that has a low-level driver to change the frequency at run time and can be chosen as default governor at compile time

CPUFreq: Ondemand Governor

Design goal

- keep the performance loss due to reduced frequency to minimum
- keep the code simple
- have tunable parameters

```
for every CPU in the system
  every X milliseconds
    get utilization since last check
    if (utilization > UP_THRESHOLD)
      increase frequency to MAX
  every Y milliseconds
    get utilization since last check
    if (utilization < DOWN_THRESHOLD)
      decrease frequency by 20%
```


CPUFreq: Ondemand Governor (cont)

Optimizations

- automatic down-scaling of frequency

  do more aggressive frequency reduction by jumping directly to the lowest frequency that can keep the CPU ~80% busy

- coordination of frequencies in software

  look at the utilization of all CPUs that are dependent, sharing the same frequency due to the hardware design, and change the frequency of all of them based on highest utilization among the group

- unify up-scaling and down-scaling paths
- parallel calculation of utilization
- dedicated workqueue

CPU Throttling

It stops the execution of commands in the CPU for certain short periods of time

- the "actual" CPU frequency is lowered
- ... but not in an homogeneous manner
- => the CPU voltage cannot be lowered in the meantime

the CPU is placed into a physical and electrical state comparable to the idling states

- throttling "forces" some "idling"
- is only useful if the CPU is less idle than the throttling rate

makes only sense if the CPU temperature has become too hot because the CPU was active excessively

- it is a good tool for "passive cooling"

on ACPI-based platforms it is user-controllable using the file "/proc/acpi/processor/*/throttling"

CPU Throttling (cont)

Throttling Rates and Power Consumption

$$P = Px\ (1 - r) + Ps\ .\ r$$

Processor[a] [b]	0 % throttling	25 % throttling	50 % throttling	75 % throttling
Mobile AMD Athlon 64 2800+	35 W	26.8 W	18.6 W	10.4 W
Intel Pentium M 1400 MHz	22 W	18.3 W	14.7 W	11.0 W

[a] The exact names of the processors are: Mobile AMD Athlon(TM) 64 Processor 2800+, Rev. CG & 1.20 V, 512 KB L2 Cache; Intel Pentium(R) M Processor, 1400 MHz & 1.484V.

[b] Data sources: AMD Athlon(TM) 64 Processor Power and Thermal Data Sheet, Publication ID: 30430, August 2004, rev. 3.37, pp. 19; Intel(R) Pentium(R) M Processor Datasheet, Order Nr. 252612, rev. 02, June 2003, pp. 11, p. 72.

Power Consumption for a specific Computing Task related to Throttling Rates

$$W = P(r)\ /\ (1 - r)$$

Processor[a]	0 % throttling	25 % throttling	50 % throttling	75 % throttling
Mobile AMD Athlon 64 2800+	35 Ws	35.7 Ws	37.2 Ws	41.6 Ws
Intel Pentium M 1400 MHz	22 Ws	24.4 Ws	29.4 Ws	44.0 Ws

[a] The exact names of the processors are: Mobile AMD Athlon(TM) 64 Processor 2800+, Rev. CG & 1.20 V, 512 KB L2 Cache; Intel Pentium(R) M Processor, 1400 MHz & 1.484V.

Outline

Power Savings Basics

CPU Power Management

- CPU Idling
- Tickless idling
- CPU Frequency Scaling
- CPU Throttling

Device and Bus power management

- backlights
- hard disks
- bus Devices (PCI, USB, PCMCIA et al.)
- device driver tuning

Applications power management

Outline

Power Savings Basics

CPU Power Management

- CPU Idling
- Tickless idling
- CPU Frequency Scaling
- CPU Throttling

Device and Bus power management

- backlights
- hard disks
- bus Devices (PCI, USB, PCMCIA et al.)
- device driver tuning

Applications power management

Biblioghraphy

- **[lesswatts]** - http://www.lesswatts.org/ a community around saving power on Linux, bringing developers, users, and sysadmins together to share software, optimizations, and tips and tricks
- [trends] – Brodowski D., *“Current trends in Linux Kernel Power Management”*
- **[tickless]** – Suresh S., Venkatesh P., Arjan V.D.V., *“Getting maximum mileage out of tickless”*, Linux Symposium 2007.
- **[cpuidle]** – Venkatesh P., Adam B., *“Do nothing, efficiently...”*, Linux Symposium 2007.
- **[ondemand]** - Venkatesh P., Alexey S., *“The Ondemand Governor”*, Linux Symposium 2006.
- **[acpi]** - The ACPI specification