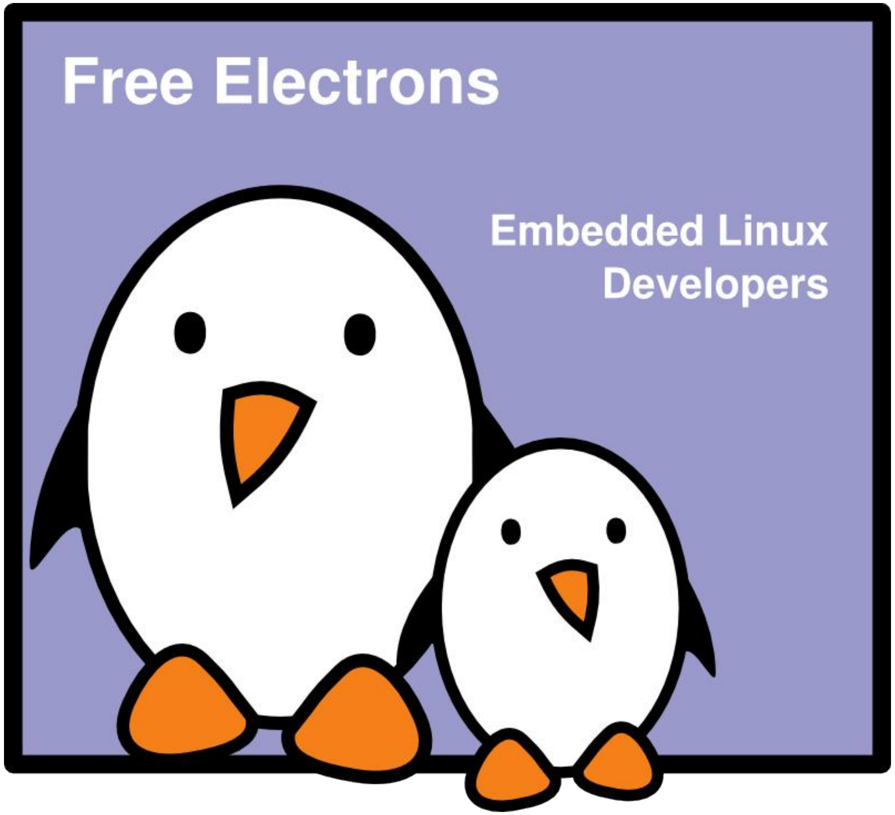
Embedded Linux optimizations

Size, RAM, speed, power, cost

Michael Opdenacker
Thomas Petazzoni
Free Electrons



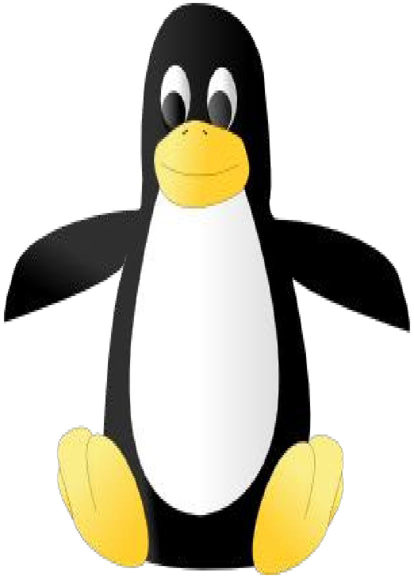
Latest update: Dec 20, 2010,
Document sources, updates and translations:
http://free-electrons.com/docs/optimizations
Corrections, suggestions, contributions and translations are welcome!

Penguin weight watchers

Make your penguin slimmer, faster, and reduce its consumption of fish!

Before

2 weeks after



CE Linux Forum

http://celinuxforum.org/

- Non profit organization, whose members are embedded Linux companies and Consumer Electronics (CE) devices makers.
- Mission: develop the use of Linux in CE devices
- Hosts many projects to improve the suitability of Linux for CE devices and embedded systems. All patches are meant to be included in the mainline Linux kernel.
- Most of the ideas introduced in this presentation have been gathered or even implemented by CE Linux Forum projects!

Contents

Ideas for optimizing the Linux kernel and executables

- Increasing speed
- Reducing size: disk footprint and RAM
- Reducing power consumption
- Global perspective: cost and combined optimization effects
- The ultimate optimization tool!

Increasing speed

Reducing kernel boot time

Measuring kernel boot time

`CONFIG_PRINTK_TIME`

- Configure it in the Kernel Hacking section.
- Adds timing information to kernel messages. Simple and robust.
- Not accurate enough on some platforms (1 jiffy = 10 ms on arm!)

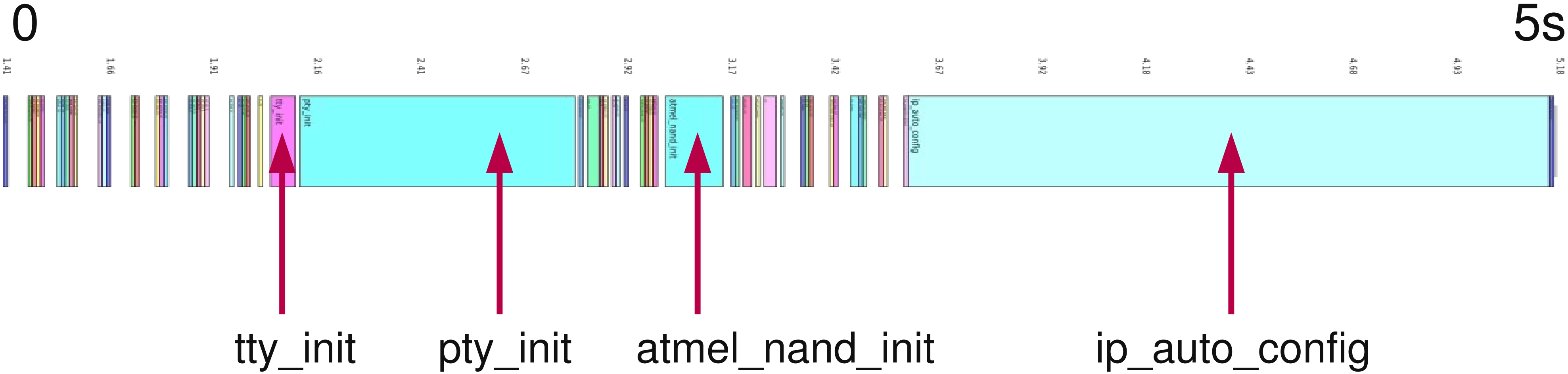
See http://elinux.org/Printk_Times

```
...
[42949372.970000] Memory: 64MB = 64MB total
[42949372.970000] Memory: 54784KB available (1404K code, 296K data, 72K init)
[42949373.180000] Mount-cache hash table entries: 512
[42949373.180000] CPU: Testing write buffer coherency: ok
[42949373.180000] checking if image is initramfs...it isn't (bad gzip magic numb
ers); looks like an initrd
[42949373.200000] Freeing initrd memory: 8192K
[42949373.210000] NET: Registered protocol family 16
...
```

Boot tracer

`CONFIG_BOOT_TRACER` in kernel configuration

- Introduced in Linux 2.6.28
  Based on the ftrace tracing infrastructure
- Allows to record the timings of initcalls
- Boot with the `initcall_debug` and `printk.time=1` parameters, run `dmesg > boot.log` and on your workstation, run
  `cat boot.log | perl scripts/bootgraph.pl > boot.svg`
  to generate a graphical representation
- Example on a board with at Atmel AT91 CPU:

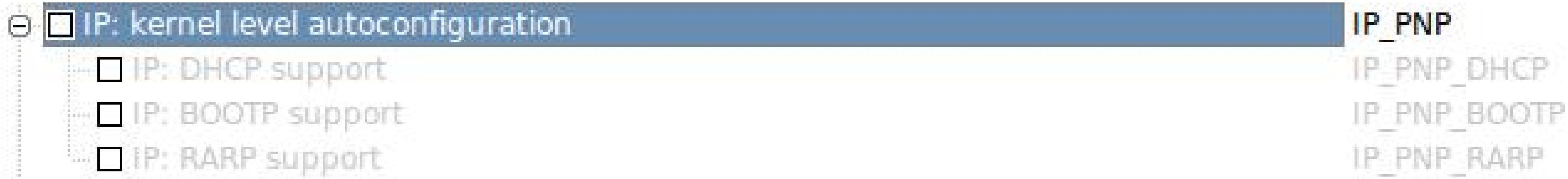


Grabserial

- From Tim Bird
  http://elinux.org/Grabserial
- A simple script to add timestamps to messages coming from a serial console.
- Key advantage: starts counting very early (bootloader), and doesn't just start when the kernel initializes.
- Another advantage: no overhead on the target, because run on the host machine.

Disable IP auto config

- Stopped initializing the IP address on the kernel command line (old remains from NFS booting, was convenient not to hardcode the IP address in the root filesystem.)
- Instead, did it in the `/etc/init.d/rcS` script.
- This saved 1.56 s on our AT91 board.
- You will save even more if you had other related options in your kernel (DHCP, BOOP, RARP)



⊖ □ IP: kernel level autoconfiguration IP_PNP
□ IP: DHCP support IP_PNP_DHCP
□ IP: BOOTP support IP_PNP_BOOTP
□ IP: RARP support IP_PNP_RARP

Reducing the number of PTYs

- PTYs are needed for remote terminals (through SSH)
  They are not needed in our dedicated system!
- The number of PTYs can be reduced through the `CONFIG_LEGACY_PTY_COUNT` kernel parameter.
  If this number is set to 4, we save 0.63 s on our Atmel board.
- As we're not using PTYs at all in our production system, we disabled them with completely with `CONFIG_LEGACY_PTYS`.
  We saved 0.64 s.
- Note that this can also be achieved without recompiling the kernel, using the `pty.legacy_count` kernel parameter.

Disable console output

- The output of kernel bootup messages to the console takes time! Even worse: scrolling up in framebuffer consoles! Console output not needed in production systems.
- Console output can be disabled with the `quiet` argument in the Linux kernel command line (bootloader settings)
- Example:

```
root=/dev/ram0 rw init=/startup.sh quiet
```

- Benchmarks: can reduce boot time by 30 or even 50%!

See http://elinux.org/Disable_Console

Preset loops_per_jiffy

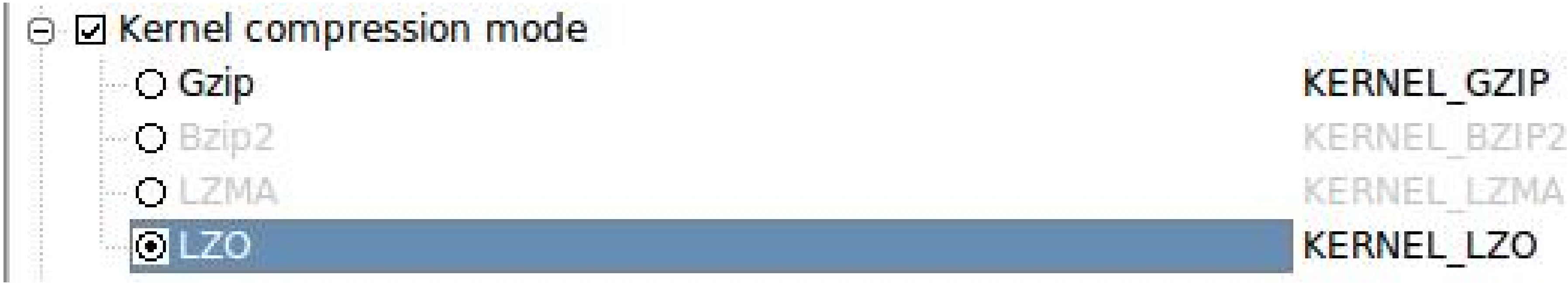
- At each boot, the Linux kernel calibrates a delay loop (for the `udelay` function). This measures a `loops_per_jiffy` (`lpj`) value. This takes about 25 jiffies (1 jiffy = time between 2 timer interrupts).
  In embedded systems, it can be about 250 ms!
- You just need to measure this once! Find the `lpj` value in kernel boot messages (if you don't get it in the console, boot Linux with the `loglevel=8` parameter). Example:

```
Calibrating using timer specific routine... 187.59
  BogoMIPS (lpj=937984)
```

- At the next boots, start Linux with the below option:
  `lpj=<value>`

LZO kernel decompression

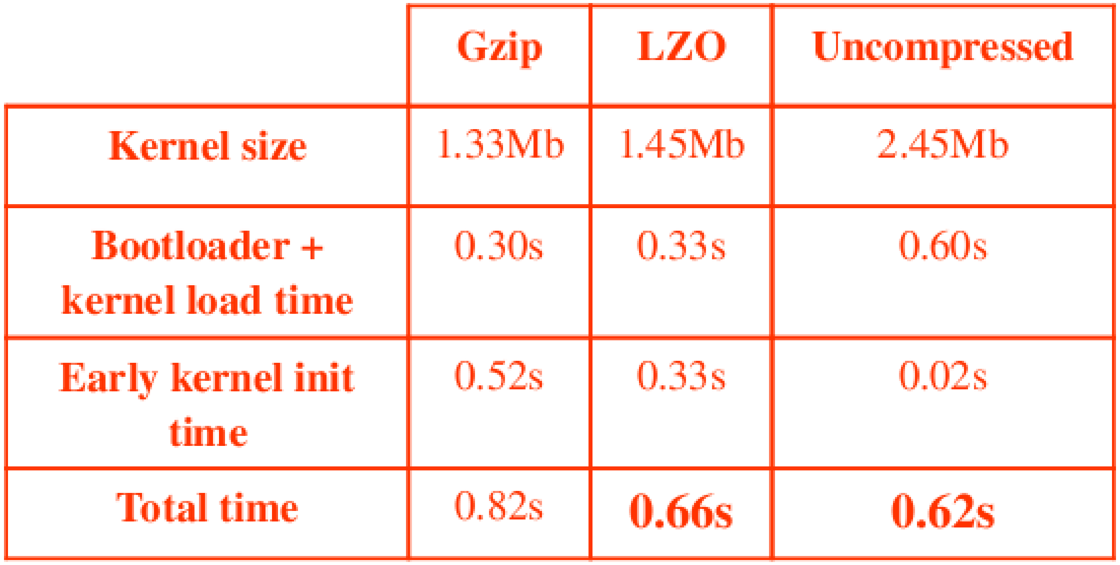
- LZO is a compression algorithm that is much faster than gzip, at the cost of a slightly degrade compression ratio (+10%).
- It was already in use in the kernel code (JFFS2, UBIFS...)
- Albin Tonnerre from Free Electrons added support for LZO compressed kernels. His patches are waiting for inclusion in mainstream Linux. Get them from http://lwn.net/Articles/350985/



⊖ ☑ Kernel compression mode
○ Gzip KERNEL_GZIP
○ Bzip2 KERNEL_BZIP2
○ LZMA KERNEL_LZMA
● LZO KERNEL_LZO

LZO decompression results

- Saves approximately 0.25 s of boot time
  See http://free-electrons.com/blog/lzo-kernel-compression/
- Our patch also allows LZO to be used for initramfs decompression (`CONFIG_INITRAMFS_COMPRESSION_LZO=y`)
- Another solution is to use an uncompressed kernel (another patch will be sent), in which case kernel execution is just marginally faster than with LZO, at the expense of a double size.



	Gzip	LZO	Uncompressed
Kernel size	1.33Mb	1.45Mb	2.45Mb
Bootloader + kernel load time	0.30s	0.33s	0.60s
Early kernel init time	0.52s	0.33s	0.02s
Total time	0.82s	**0.66s**	**0.62s**

Directly boot Linux from bootstrap code

- Idea: make a slight change to at91bootstrap to directly load and execute the Linux kernel image instead of the U-boot one.
- Rather straightforward when boot U-boot and the kernel are loaded from NAND flash.
- Requires to hardcode the kernel command line in the kernel image (`CONFIG_CMDLINE`)
- Requires more development work when U-boot is loaded from a different type of storage (SPI dataflash, for example).
  In this case, you can keep U-boot, but remove all the features not needed in production (USB, Ethernet, tftp...)
- Time savings: about 2 s

See http://free-electrons.com/blog/at91bootstrap-linux/

Reduce the kernel size

Through the `CONFIG_EMBEDDED` option

- Remove things that are not needed in your dedicated system (features, debugging facilities and messages)
- Make sure you have no unused kernel drivers
- Disable support for loadable kernel modules and make all your drivers static (unless there are multiple drivers than can be loaded later).
- A smaller kernel is faster to load
- A simpler kernel executes faster
- At least, compile drivers as modules for devices not used at boot time. This reduces time spent initializing drivers.

Faster rebooting (1)

`kexec` system call: executes a new kernel from a running one.

- Must faster rebooting: doesn't go through bootstrap / bootloader code.
- Great solution for rebooting after system (“firmware”) upgrades.
- Useful for automatic rebooting after kernel panics.

See http://developer.osdl.org/andyp/kexec/whitepaper/kexec.pdf and Documentation/kdump/kdump.txt in kernel sources.

Faster rebooting (2)

Another option: use `reboot=soft` in the kernel command line

- When you reboot, the firmware will be skipped.
- Drawback: unlike `kexec`, cannot be chosen from userspace.
- Supported platforms: `i386`, `x86_64`, `arm`, `arm26` (Aug. 2006)
- See `Documentation/kernel-parameters.txt` in the kernel sources for details. Not supported on all platforms.

Skip memory allocation

Idea: spare memory at boot time and manage it by yourself!

- Assume you have 32 MB of RAM
- Boot your kernel with `mem=30`
  The kernel will just manage the first 30 MB of RAM.
- Driver code can now reclaim the 2 MB left:

```
buf = ioremap (
           0x1e00000,       /* Start: 30 MB */
           0x200000         /* Size: 2 MB */
              );
```

- This saves time allocating memory.
  Critical drivers are also sure to always have the RAM they need.

Kernel boot time - Other ideas

- Copy kernel and initramfs from flash to RAM using DMA
  (Used by MontaVista in Dell Latitude ON)
- Fast boot, asynchronous initcalls: http://lwn.net/Articles/314808/
  Mainlined, but API still used by very few drivers.
  Mostly useful when your CPU has idle time in the boot process.
- Use deferred initcalls
  See http://elinux.org/Deferred_Initcalls
- NAND: just check for bad blocks once
  Atmel: see http://patchwork.ozlabs.org/patch/27652/

See http://elinux.org/Boot_Time for more resources

Increasing speed

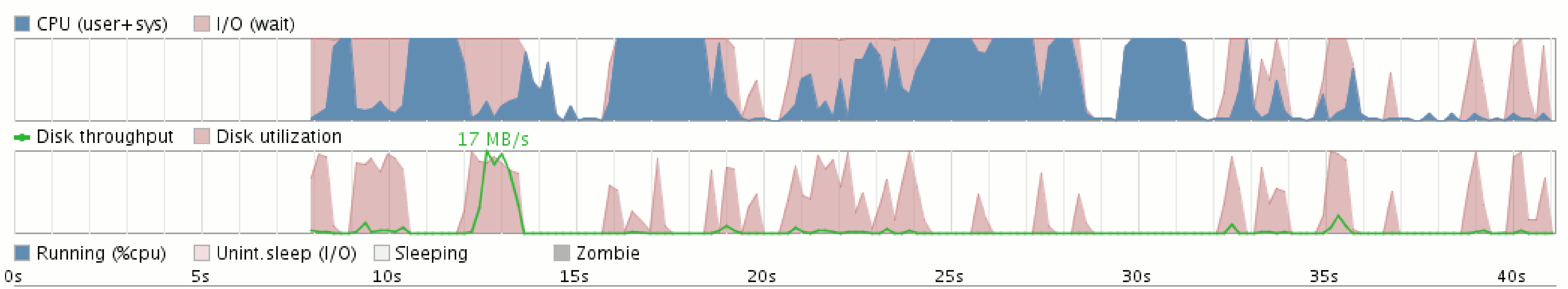
System startup time and application speed

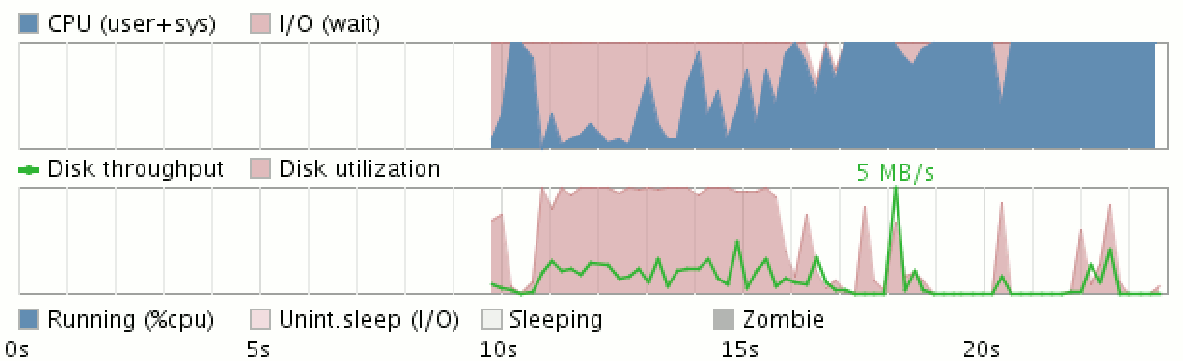
Starting system services

- SysV `init`:
  Starts services sequentially. Waits for the current startup script to be complete to start the next one! While dependencies exist, some tasks can be run in parallel!

- Initng: http://initng.org
  New alternative to SysV init, which can start services in parallel, as soon as their preconditions are met.

Initng vs. SysV init: bootcharts





Initng wins!
System utilization is much better.

- You can hunt system startup trouble by using the Bootchart program (http://www.bootchart.org/).
- Bootchart is slow (Java) and not very accurate.
  See http://elinux.org/Bootchart for solutions for embedded systems.

Reading ahead

- Linux keeps the contents of all the files it reads in RAM (in the *page cache*), as long as it doesn't need the RAM pages for something else.
- Idea: load files (programs and libraries in particular) in RAM cache before using them. Best done when the system is not doing any I/O.
- Thanks to this, programs are not stuck waiting for I/O.
  Used the Knoppix distribution to achieve very nice boot speed-ups.
- Also planned to be used by Initng.
- Not very useful for systems with very little RAM:
  cached pages are recycled before the files are accessed.

Implementing readahead

- You can use the `sys_readahead()` system call in your C programs. See `man readahead` for details.
- You can also use the `readahead-list` utility, which reads a file containing the list of files to load in cache. Available on: http://freshmeat.net/projects/readahead-list/.
- In embedded systems using Busybox, you can use the `readahead` command (implemented by Free Electrons).

Compiler speed optimizations

- By default, most tools are compiled with compiler optimizations. Make sure you use them for your own programs!
- `-O2` is the most common optimization switch of `gcc`.
  Lots of optimization techniques are available.
  See http://en.wikipedia.org/wiki/Compiler_optimization
- `-O3` can be also be used for speed critical executables.
  However, there is done at the expense of code size (for example “inlining”: replacing function calls by the function code itself).

Using processor acceleration instructions

- liboil - http://liboil.freedesktop.org/
  Library of functions optimized for special instructions from several processors (Altivec, MMX, SSE, etc.)
- Mainly functions implementing loops on data arrays:
  type conversion, copying, simple arithmetics, direct cosine transform, random number generation...
- Transparent: keeps your application portable!
- So far mainly supports desktop processors
- License: BSD type

Prelinking (1)

Applies to executables using shared libraries

- To load and start an executable, the dynamic linker has a significant amount of work to do (mainly address relocation)
- It can take a lot of time for executables using many shared libraries!
- In many systems in which executables and shared libraries never change, the same job is done every time the executable is started.

Prelinking (2)

`prelink`
http://people.redhat.com/jakub/prelink/

- `prelink` modifies executables and shared libraries to simplify the dynamic linker relocation work.
- This can greatly reduce startup time for big applications (50% less for KDE!). This also saves memory consumed by relocations.
- Can be used to reduce the startup time of a Linux system.
- Just needs to be run again when libraries or executables are updated.

Details on http://elinux.org/Pre_Linking

Use simpler Unix executables

- Big, feature rich executables take time to load.
  Particularly true for shell scripts calling the `bash` shell!
- Idea: replace standard Unix / GNU executables by lightweight, simplified implementations by busybox (http://busybox.net).
- Implemented by Ubuntu 6.10 to reduce boot time, replacing `bash` (649 K) by `dash` (79 K, see http://en.wikipedia.org/wiki/Debian_Almquist_shell).
  This broke various shell scripts which used bash specific features (“bashisms”).
- In non-embedded Linux systems
  where feature-rich executables are still needed,
  should at least use `busybox ash` for system scripts.

Shells: reducing forking

- `fork` / `exec` system calls are very heavy.
  Because of this, calls to executables from shells are slow.
- Even executing `echo` in busybox shells results in a `fork` syscall!
- Select `Shells -> Standalone shell` in busybox configuration to make the busybox shell call applets whenever possible.
- Pipes and back-quotes are also implemented by `fork` / `exec`.
  You can reduce their usage in scripts. Example:
  `cat /proc/cpuinfo | grep model`
  Replace it with: `grep model /proc/cpuinfo`

See http://elinux.org/Optimize_RC_Scripts

Use faster filesystems

Run faster by using the most appropriate filesystems!

- Compressed read-only filesystem (block device): use SquashFS (http://squashfs.sourceforge.net) instead of CramFS (much slower, getting obsolete).
- NAND flash storage: you should try UBIFS (http://www.linux-mtd.infradead.org/doc/ubifs.html), the successor of JFFS2. It is much faster. You could also use SquashFS. See our Choosing filesystems presentation (http://free-electrons.com/docs/filesystems).

Use faster filesystems (2)

- Use RAM filesystems for temporary, speed critical files with no need for permanent storage. Details in the kernel sources: `Documentation/filesystems/tmpfs.txt`
- Benchmark your system and application on competing filesystems! Reiser4 is more innovative and benchmarks found it faster than ext3.
- Good to benchmark your system with JFS or XFS too. XFS is reported to be the fastest to mount (good for startup time), and JFS to have the lowest CPU utilization. See http://www.debian-administration.org/articles/388
- ext4 is also ready to be used now.

Speed up applications with tmpfs

- When enough RAM is available, the OS keeps recently accessed files and applications in RAM (*page cache*). This significantly speeds up any new usage. However, depending on system activity, this may not last long.
- For programs that need fast startup even if they haven't been run for a long time: copy them to a tmpfs filesystem at system startup! This makes sure they are always accessed from the file cache in RAM (provided you do not have a swap partition).
- See Documentation/filesystems/tmpfs.txt in kernel sources for details about tmpfs.
- Caution: don't use ramdisks instead!
  Ramdisks duplicate files in RAM and unused space cannot be reclaimed.
- Caution: use with care. May impact overall performance.
  Not needed if there's enough RAM to cache all files and programs.

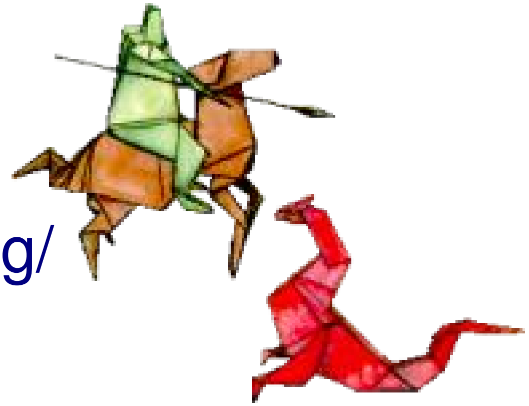
Boot from a hibernate image

The ultimate technique for instant boot!

- In development: start the system, required applications and the user interface. Hibernate the system to disk / flash in this state.
- In production: boot the kernel and restore the system state from with this predefined hibernation image.
- This way, you don't have to initialize the programs one by one. You just get the back to a valid state.
- Used in Sony cameras to achieve instant power on time.
- Unlike Suspend to RAM, still allows to remove batteries!

Use a profiler

- Using a profiler can help to identify unexpected behavior degrading application performance.
- For example, a profiler can tell you in which functions most of the time is spent.
- Possible to start with strace and ltrace
- Advanced profiling with Valgrind: http://valgrind.org/
  - Compile your application for `x86` architecture
  - You can then profile it with the whole Valgrind toolsuite:
    Cachegrind: sources of cache misses and function statistics.
    Massif: sources of memory allocation.
- See our Software Development presentation for details:
  http://free-electrons.com/docs/swdev/



Reducing size

Kernel size and RAM usage

Linux-Tiny

Goal: reduce the disk footprint and RAM size of the Linux kernel
http://elinux.org/Linux_Tiny

- Set of patches against the mainstream Linux kernel.
  Mergeability in mainstream is a priority.
  Many changes have already been merged in recent kernels.
- All features can be selected in kernel configuration (`CONFIG_EMBEDDED`).
- Also ships utilities or patches for tracking sources of memory usage or code size.

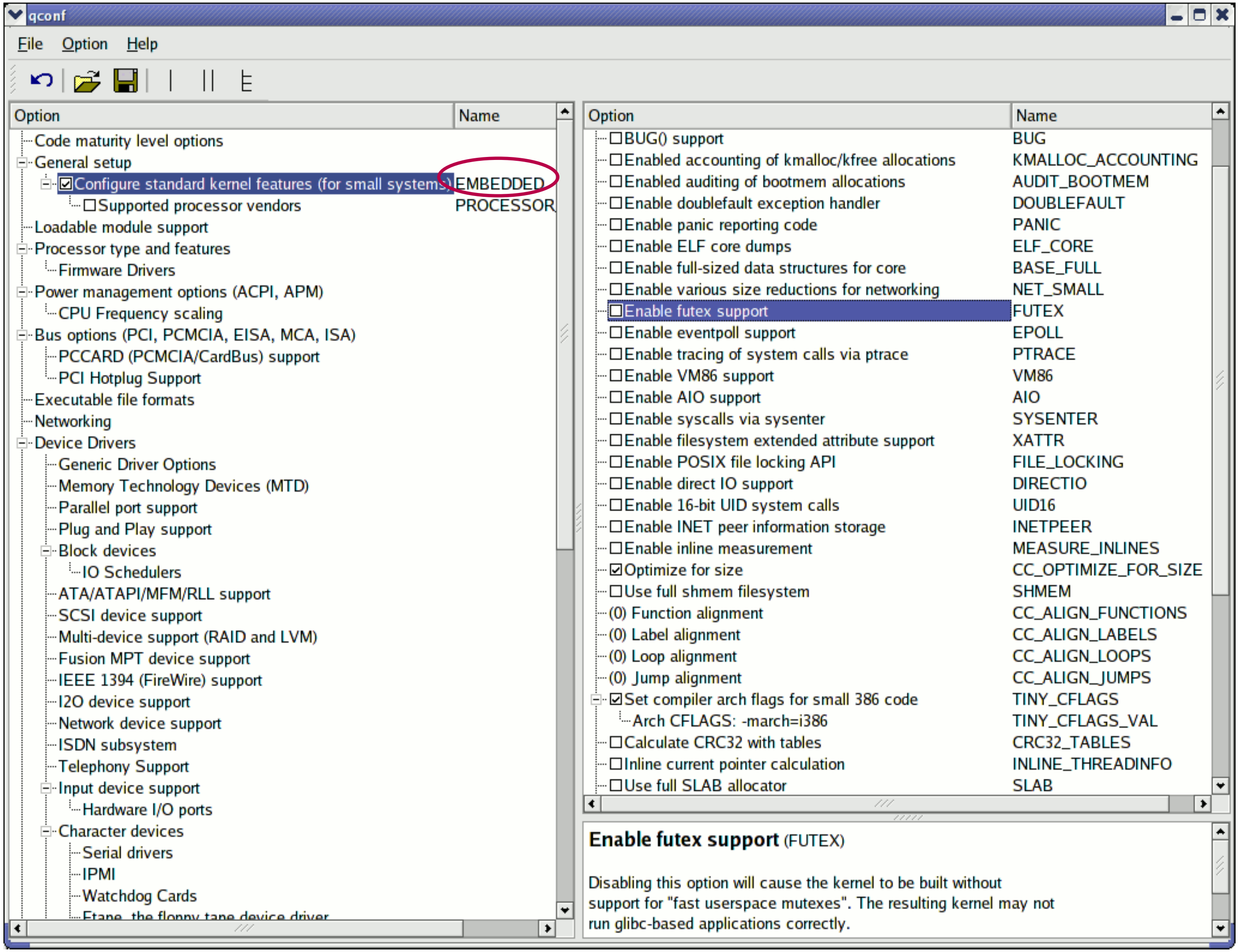
Linux-Tiny ideas (1)

- Remove kernel messages (`printk`, `BUG`, `panic`...)
- Hunt excess inlining (speed vs. size tradeoff)
  2.6.26: can allow gcc to uninline functions marked as inline: (`CONFIG_OPTIMIZE_INLINING=y`). Only used by `x86` so far.
- Hunt excess memory allocations
- Memory (slob instead of slab) allocator more space efficient for small systems.
- Reduce the size of kernel data structures (may impact performance)
- Simpler alternative implementations of kernel functionalities with less features, or not supporting special cases.

Linux-Tiny ideas (2)

- Remove some features which may not be needed in some systems.
- Compiling optimizations for size.
- A smaller kernel executable also saves RAM (unless executed in place from storage).

Linux-Tiny: kernel configuration screenshot



Option	Name
Code maturity level options	
General setup	
☑ Configure standard kernel features (for small systems)	EMBEDDED
□ Supported processor vendors	PROCESSOR
Loadable module support	
Processor type and features	
Firmware Drivers	
Power management options (ACPI, APM)	
CPU Frequency scaling	
Bus options (PCI, PCMCIA, EISA, MCA, ISA)	
PCCARD (PCMCIA/CardBus) support	
PCI Hotplug Support	
Executable file formats	
Networking	
Device Drivers	
Generic Driver Options	
Memory Technology Devices (MTD)	
Parallel port support	
Plug and Play support	
Block devices	
IO Schedulers	
ATA/ATAPI/MFM/RLL support	
SCSI device support	
Multi-device support (RAID and LVM)	
Fusion MPT device support	
IEEE 1394 (FireWire) support	
I2O device support	
Network device support	
ISDN subsystem	
Telephony Support	
Input device support	
Hardware I/O ports	
Character devices	
Serial drivers	
IPMI	
Watchdog Cards	
Ftape, the floppy tape device driver	

Option	Name
□ BUG() support	BUG
□ Enabled accounting of kmalloc/kfree allocations	KMALLOC_ACCOUNTING
□ Enabled auditing of bootmem allocations	AUDIT_BOOTMEM
□ Enable doublefault exception handler	DOUBLEFAULT
□ Enable panic reporting code	PANIC
□ Enable ELF core dumps	ELF_CORE
□ Enable full-sized data structures for core	BASE_FULL
□ Enable various size reductions for networking	NET_SMALL
□ Enable futex support	FUTEX
□ Enable eventpoll support	EPOLL
□ Enable tracing of system calls via ptrace	PTRACE
□ Enable VM86 support	VM86
□ Enable AIO support	AIO
□ Enable syscalls via sysenter	SYSENTER
□ Enable filesystem extended attribute support	XATTR
□ Enable POSIX file locking API	FILE_LOCKING
□ Enable direct IO support	DIRECTIO
□ Enable 16-bit UID system calls	UID16
□ Enable INET peer information storage	INETPEER
□ Enable inline measurement	MEASURE_INLINES
☑ Optimize for size	CC_OPTIMIZE_FOR_SIZE
□ Use full shmem filesystem	SHMEM
(0) Function alignment	CC_ALIGN_FUNCTIONS
(0) Label alignment	CC_ALIGN_LABELS
(0) Loop alignment	CC_ALIGN_LOOPS
(0) Jump alignment	CC_ALIGN_JUMPS
☑ Set compiler arch flags for small 386 code	TINY_CFLAGS
Arch CFLAGS: -march=i386	TINY_CFLAGS_VAL
□ Calculate CRC32 with tables	CRC32_TABLES
□ Inline current pointer calculation	INLINE_THREADINFO
□ Use full SLAB allocator	SLAB

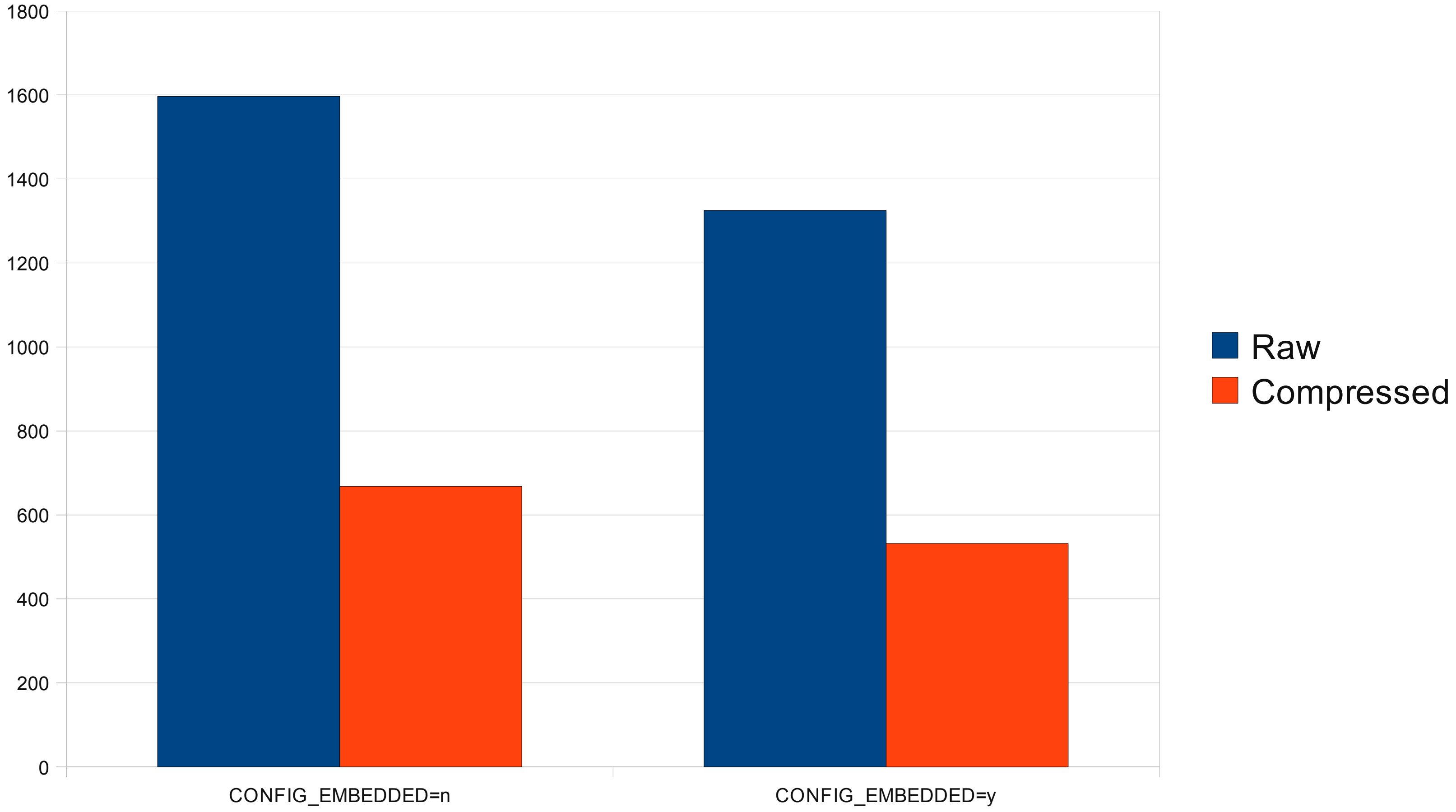
Enable futex support (FUTEX)

Disabling this option will cause the kernel to be built without support for "fast userspace mutexes". The resulting kernel may not run glibc-based applications correctly.

Many features configured out

With and without CONFIG_EMBEDDED

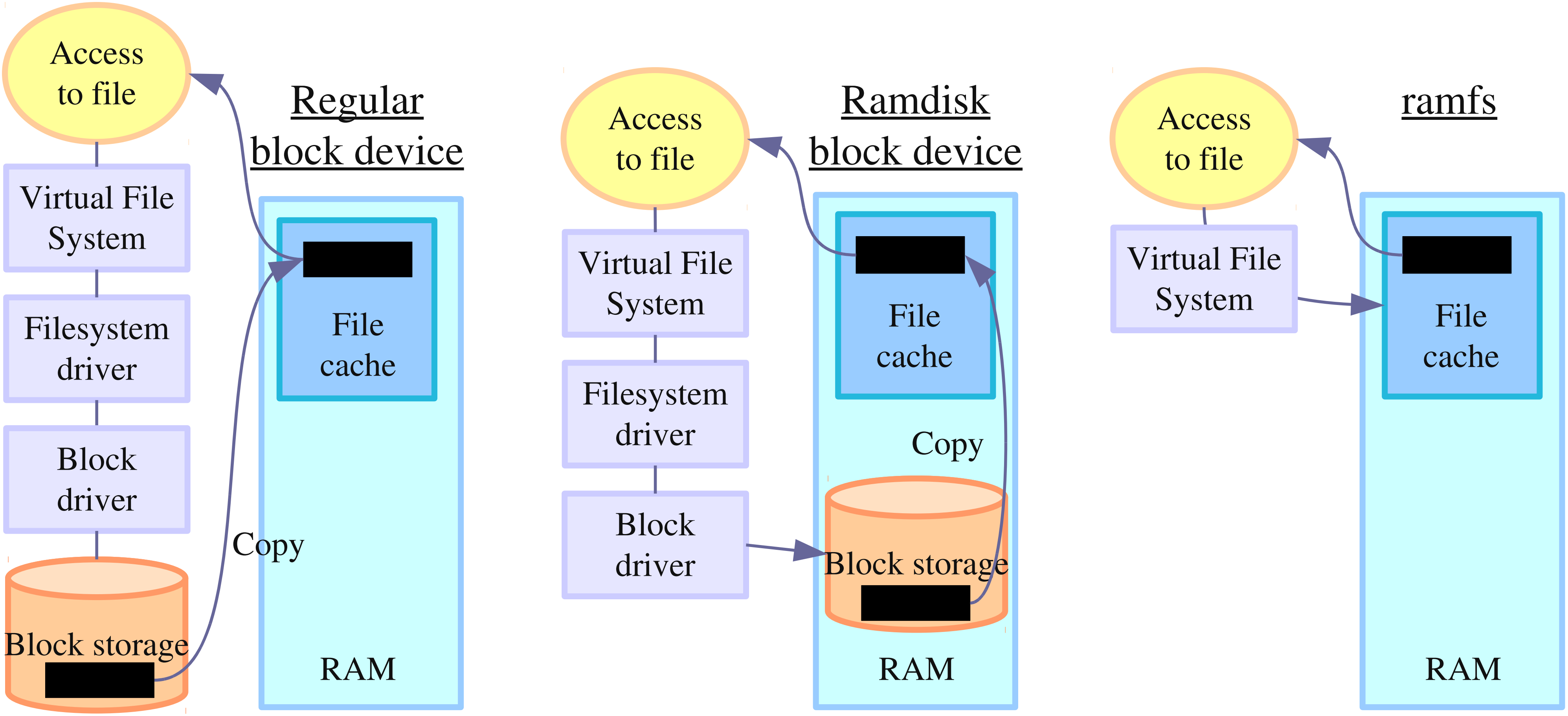
Tests on Linux 2.6.29, on a minimalistic but working x86 kernel



Raw: -272 KB (-17%), Compressed: -136 KB (-20%)

Replace initrd by initramfs

Replace init ramdisks (initrd) with initramfs:
much less overhead and ram waste!



ramfs advantages over ramdisks

- No block and filesystem overhead.
- No duplication in RAM.
- Files can be removed (reclaiming RAM) after use.
- Initramfs: ramfs archive embedded in the Linux kernel file.

Reducing size

Application size and RAM usage

Static or dynamic linking? (1)

Static linking

- ▶ All shared library code duplicated in the executables
- 🟢 Allows not to copy the C library in the filesystem.
  Simpler and smaller when very few executables (busybox)
- 🔴 Library code duplication: bad for systems with more executables (code size and RAM)

Best for small systems (< 1-2 MB) with few executables!

Static or dynamic linking? (2)

Dynamic linking

- Shared library code not duplicated in the executables
- Makes much smaller executables
- Saves space in RAM (bigger executables take more RAM)
- Requires the library to the copied to the filesystem

Best for medium to big systems (> 500 KB - 1 MB)

Using a lighter C library

- glibc (GNU C library): http://www.gnu.org/software/libc/
  Found on most computer type GNU/Linux machines
  Size on `arm`: approx 1.7 MB
- uClibc: http://www.uclibc.org/
  Found in more and more embedded Linux systems!
  Size on `arm`: approx 400 KB (you save 1.2 MB!)
- Executables are slightly smaller too:

C program	*Compiled with shared libraries*		*Compiled statically*	
	glibc	*uClibc*	*glibc*	*uClibc*
Plain "hello world"	4.6 K	4.4 K	475 K	25 K
Busybox	245 K	231 K	843 K	311 K

How to use uClibc?

- Need to compile all your executables with a uClibc toolchain.
- Ready-to-use toolchains can be found on http://free-electrons.com/community/tools/uclibc
- You can very easily build your own with buildroot: http://buildroot.uclibc.org/
- You also have to copy the uClibc files from the toolchain to the `/lib` directory in the target root filesystem.
- Ready-to-use filesystems can also be generated by buildroot. You just need to add your specific stuff then.

Need for stripping

- Compiled executables and libraries contain extra information which can be used to investigate problems in a debugger.
- This was useful for the tool developer, but not for the final user.
- To remove debugging information, use the `strip` command. This can save a very significant amount of space!

```
gcc -o hello hello.c    (output size: 4635 bytes)
strip hello             (output size: 2852 bytes, -38.5%)
```

- Don't forget to strip libraries too!

Are my executables stripped?

You can use the `file` command to get the answer

```
gcc -o hello hello.c
file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses
shared libs), not stripped
```

```
strip hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), for GNU/Linux 2.2.5, dynamically linked (uses
shared libs), stripped
```

You can use `findstrip` (http://packages.debian.org/stable/source/perforate) to find all executables and libraries that need stripping in your system.

How to strip

- Some lightweight tools, like busybox, are automatically stripped when you build them.
- Makefiles for many standard tools offer a special command:
  `make install-strip`
- Caution: stripping is architecture dependent.
  Use the strip command from your cross-compiling toolchain:
  `arm-linux-strip potato`

sstrip: “super strip”

http://muppetlabs.com/~breadbox/software/elfkickers.html

- Goes beyond strip and can strip out a few more bits that are not used by Linux to start an executable.
- Can be used on libraries too. Minor limitation: processed libraries can no longer be used to compile new executables.
- Can also be found in toolchains made by Buildroot (optional)

	Hello World	***Busybox***	***Inkscape***
Regular	4691 B	287783 B	11397 KB
stripped	2904 B (-38 %)	230408 B (-19.9 %)	9467 KB (-16.9 %)
sstripped	1392 B (-70 %)	229701 B (-20.2 %)	9436 KB (-17.2 %)

Best for tiny executables!

Library Optimizer

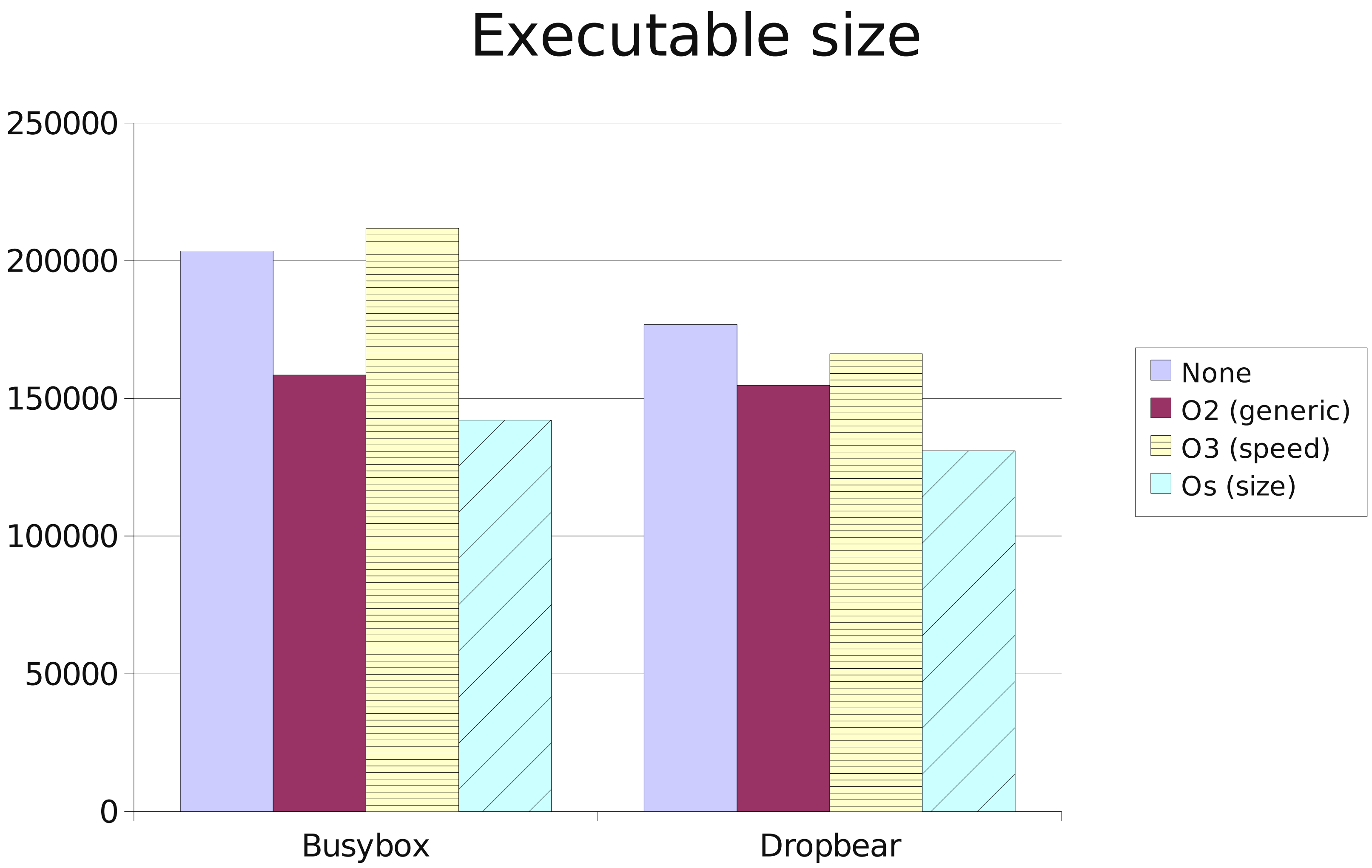
http://libraryopt.sourceforge.net/

- Contributed by MontaVista
- Examines the complete target file system, resolves all shared library symbol references, and rebuilds the shared libraries with only the object files required to satisfy the symbol references.
- Can also take care of stripping executables and libraries.
- However, requires to rebuild all the components from source. Would be nicer to achieve this only with ELF manipulations.

Compiler space optimizations

- Regular compiler optimizations simplifying code also reduce size
- You can also reduce the size of executables by asking `gcc` to optimize generated code size:
  `gcc -Os -o husband husband.c`
- `-Os` corresponds to `-O2` optimizations except the ones increasing size, plus extra size-specific ones.
- `-Os` is already used by default to build busybox.
- Possible to further reduce the size by compiling and optimizing all sources at once, with the `-fwhole-program --combine` gcc options.
- See http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html for all gcc optimization options.

Simple gcc optimization benchmark



Restartable applications

- When RAM is scarce, can be useful to abort applications that are not in use (for example hidden graphical interfaces).
- Better to do it before the Linux Kernel OOM (Out Of Memory) killer comes and makes bad decisions.
- You can use the “Linux Checkpoint / Restart” project to have the Linux kernel save the state of a running application so that it can later resume its execution from the time at which it was checkpointed.

  See http://www.linux-cr.org/ for details.

Compressing filesystems

Can significantly increase your storage capacity

- MTD (flash or ROM) storage: use UBIFS or JFFS2 for small partitions.
- Block storage: use SquashFS (http://squashfs.sourceforge.net) instead of CramFS for read-only partitions. It compresses much better and is much faster too.

Merging duplicate files

Software compiling and installing often create duplicate files...
Check that your root filesystem doesn’t contain any!

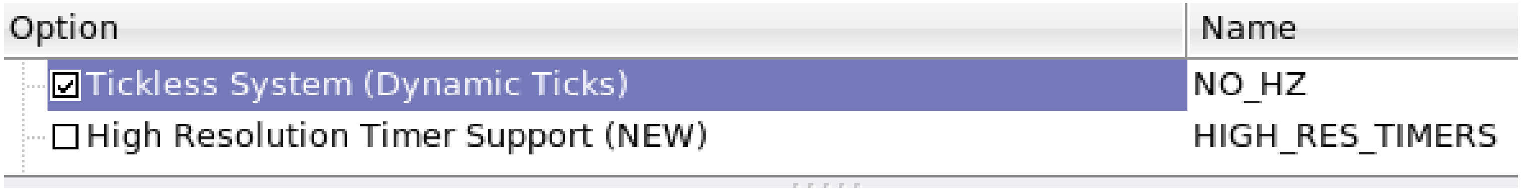
- dupmerge2: http://sourceforge.net/projects/dupmerge
  Replaces duplicate files by hard links.
- clink: http://free-electrons.com/community/tools/utils/clink
  Replaces duplicate files by symbolic links.
  Example: saves 4% of total space in Fedora Core 5.
- finddup: http://www.shelldorado.com/scripts/cmds/finddup
  Finds duplicate files.

Reducing power consumption

Tickless kernel

Kernel configuration: `NO_HZ` setting in Processor type and features

- To implement multitasking, the processor receives a timer interrupt at a given frequency (every 4 ms by default on Linux 2.6). On idle systems, this wakes up the processor all the time, just to realize there is nothing to do!
- Idea: when all processors are idle, disable the timer interrupt, and re-enable it when something happens (a real interrupt). This saves power in laptops, in embedded systems and with virtual servers!
- 2.6.24: supports `x86`, `arm`, `mips` and `powerpc`

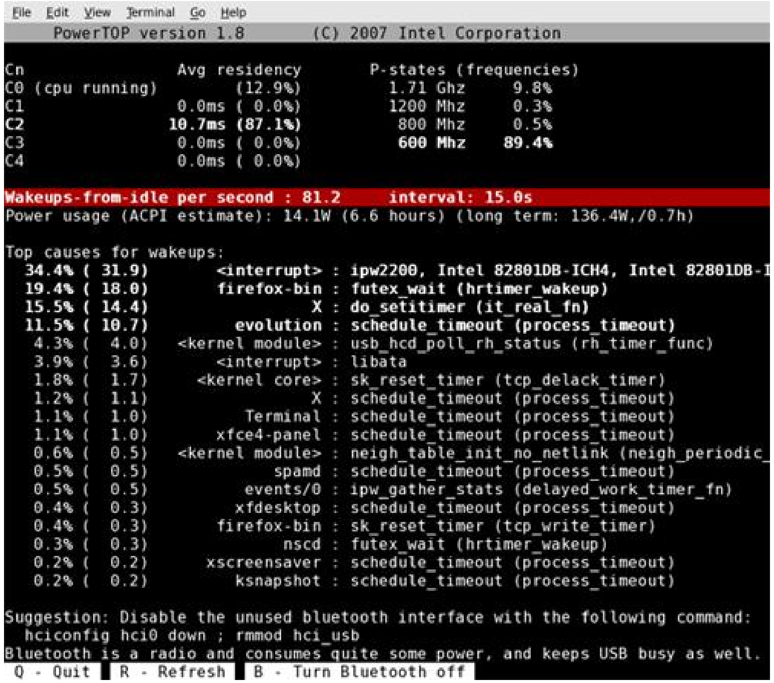


Option	Name
☑ Tickless System (Dynamic Ticks)	NO_HZ
□ High Resolution Timer Support (NEW)	HIGH_RES_TIMERS

PowerTOP

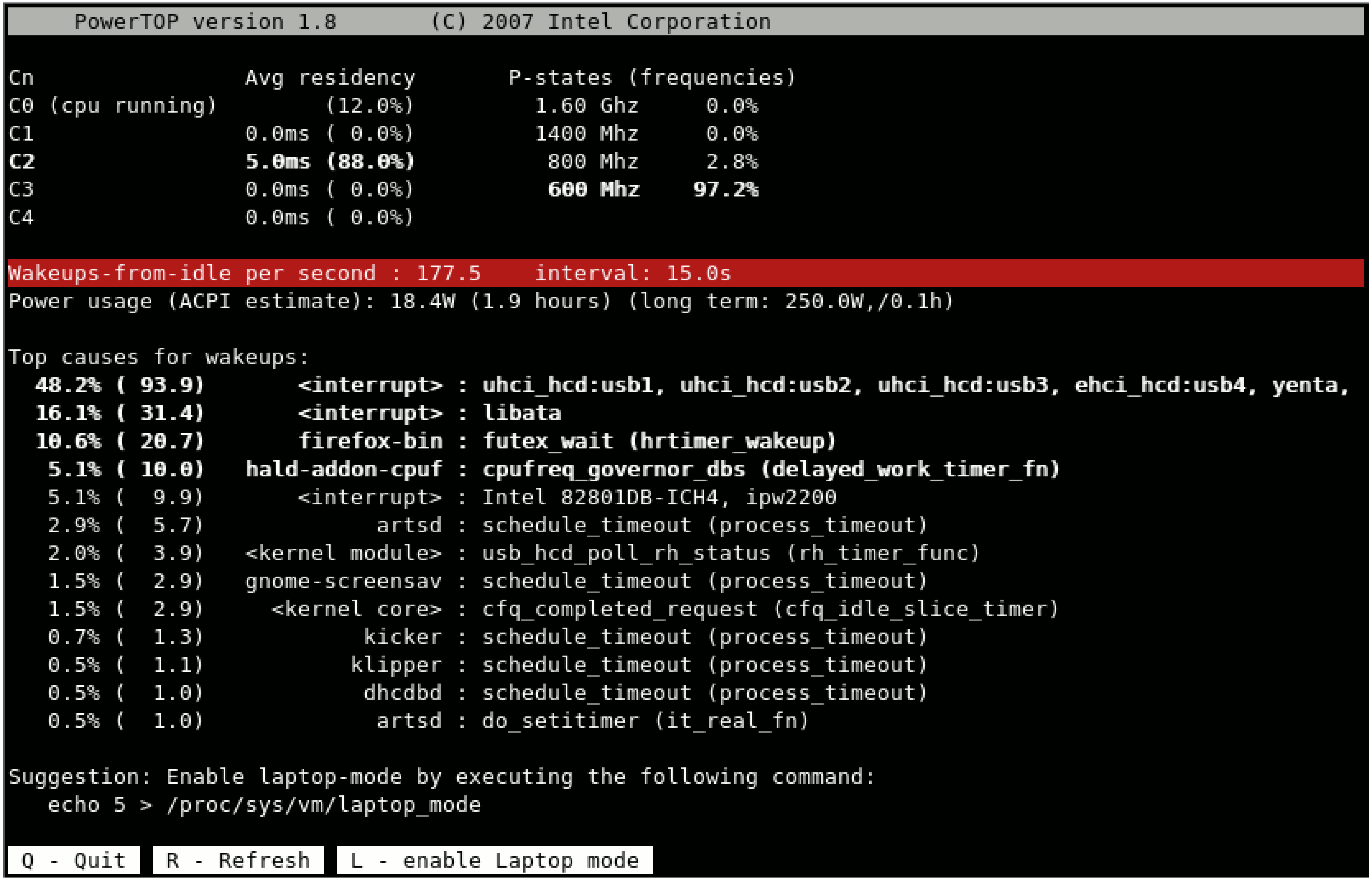
http://www.lesswatts.org/projects/powertop/

- With dynamic ticks, allows to fix parts of kernel code and applications that wake up the system too often.
- PowerTOP allows to track the worst offenders
- Now available on ARM cpus implementing CPUidle
- Also gives you useful hints for reducing power.



```
     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
```

PowerTOP in action



```
     PowerTOP version 1.8       (C) 2007 Intel Corporation

Cn                  Avg residency       P-states (frequencies)
C0 (cpu running)          (12.0%)         1.60 Ghz     0.0%
C1                  0.0ms ( 0.0%)         1400 Mhz     0.0%
C2                  5.0ms (88.0%)          800 Mhz     2.8%
C3                  0.0ms ( 0.0%)          600 Mhz    97.2%
C4                  0.0ms ( 0.0%)

Wakeups-from-idle per second : 177.5    interval: 15.0s
Power usage (ACPI estimate): 18.4W (1.9 hours) (long term: 250.0W,/0.1h)

Top causes for wakeups:
  48.2% ( 93.9)       <interrupt> : uhci_hcd:usb1, uhci_hcd:usb2, uhci_hcd:usb3, ehci_hcd:usb4, yenta,
  16.1% ( 31.4)       <interrupt> : libata
  10.6% ( 20.7)       firefox-bin : futex_wait (hrtimer_wakeup)
   5.1% ( 10.0)   hald-addon-cpuf : cpufreq_governor_dbs (delayed_work_timer_fn)
   5.1% (  9.9)       <interrupt> : Intel 82801DB-ICH4, ipw2200
   2.9% (  5.7)             artsd : schedule_timeout (process_timeout)
   2.0% (  3.9)   <kernel module> : usb_hcd_poll_rh_status (rh_timer_func)
   1.5% (  2.9)   gnome-screensav : schedule_timeout (process_timeout)
   1.5% (  2.9)     <kernel core> : cfq_completed_request (cfq_idle_slice_timer)
   0.7% (  1.3)            kicker : schedule_timeout (process_timeout)
   0.5% (  1.1)           klipper : schedule_timeout (process_timeout)
   0.5% (  1.0)            dhcdbd : schedule_timeout (process_timeout)
   0.5% (  1.0)             artsd : do_setitimer (it_real_fn)

Suggestion: Enable laptop-mode by executing the following command:
   echo 5 > /proc/sys/vm/laptop_mode

 Q - Quit    R - Refresh    L - enable Laptop mode
```

cpufreq

Configuration: `CPU_FREQ` in Power management options

- Allows to change the CPU frequency on the fly
- Supported architectures (2.6.20):
  `i386`, `sh`, `ia64`, `sparc64`, `x86_64`, `powerpc`, `arm` (`i.MX` only).
- Usually controlled from userspace through `/sys` by a user configurable *governor* process, according to CPU load, heat, battery status... The most common is `cpuspeed`:
  http://carlthompson.net/software/cpuspeed/
- Saves a significant amount of battery life in notebooks.

Suspend hidden GUIs

Idea: suspend hidden user interfaces to save CPU and power.

- Send a suspend (stop) signal:
  ```
  kill -SIGTSTP <pid>
  ```
- Send a continue signal:
  ```
  kill -SIGCONT <pid>
  ```

Software suspend

http://www.suspend2.net/

- Lots of great features for notebook users, such as RAM suspend or hibernate to disk.
- Unfortunately, restricted on some Intel compatible processors and targeting only machines with APM or ACPI (rarely found in non PC embedded systems!).
- Not addressing the requirements of embedded systems (support for other CPUs, voltage reduction...).

Power management resources

- http://free-electrons.com/docs/power/
  Our presentation on power management in the Linux kernel
  What you need to implement in your BSP and device drivers.
- http://lesswatts.org
  Intel effort trying to create a Linux power saving community.
  Mainly targets Intel processors.
  Lots of useful resources.
- http://wiki.linaro.org/WorkingGroups/PowerManagement/
  Ongoing development on the ARM platform.
- Tips and ideas for prolonging battery life:
  http://j.mp/fVdxKh

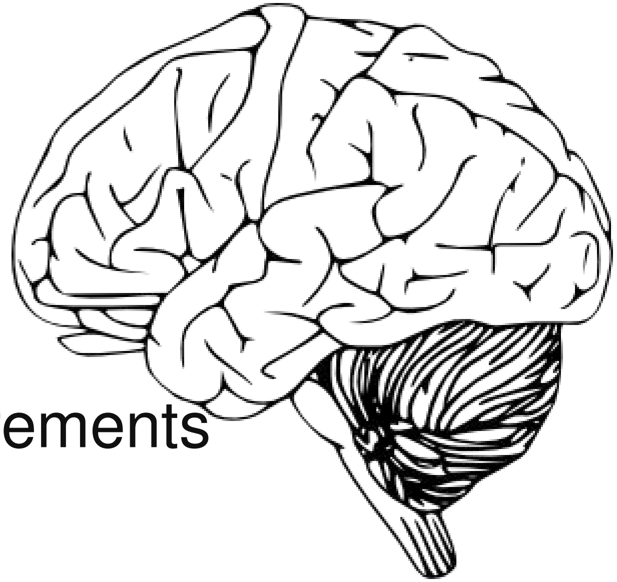
Global perspective

Cost and combined optimization effects

Combined benefits

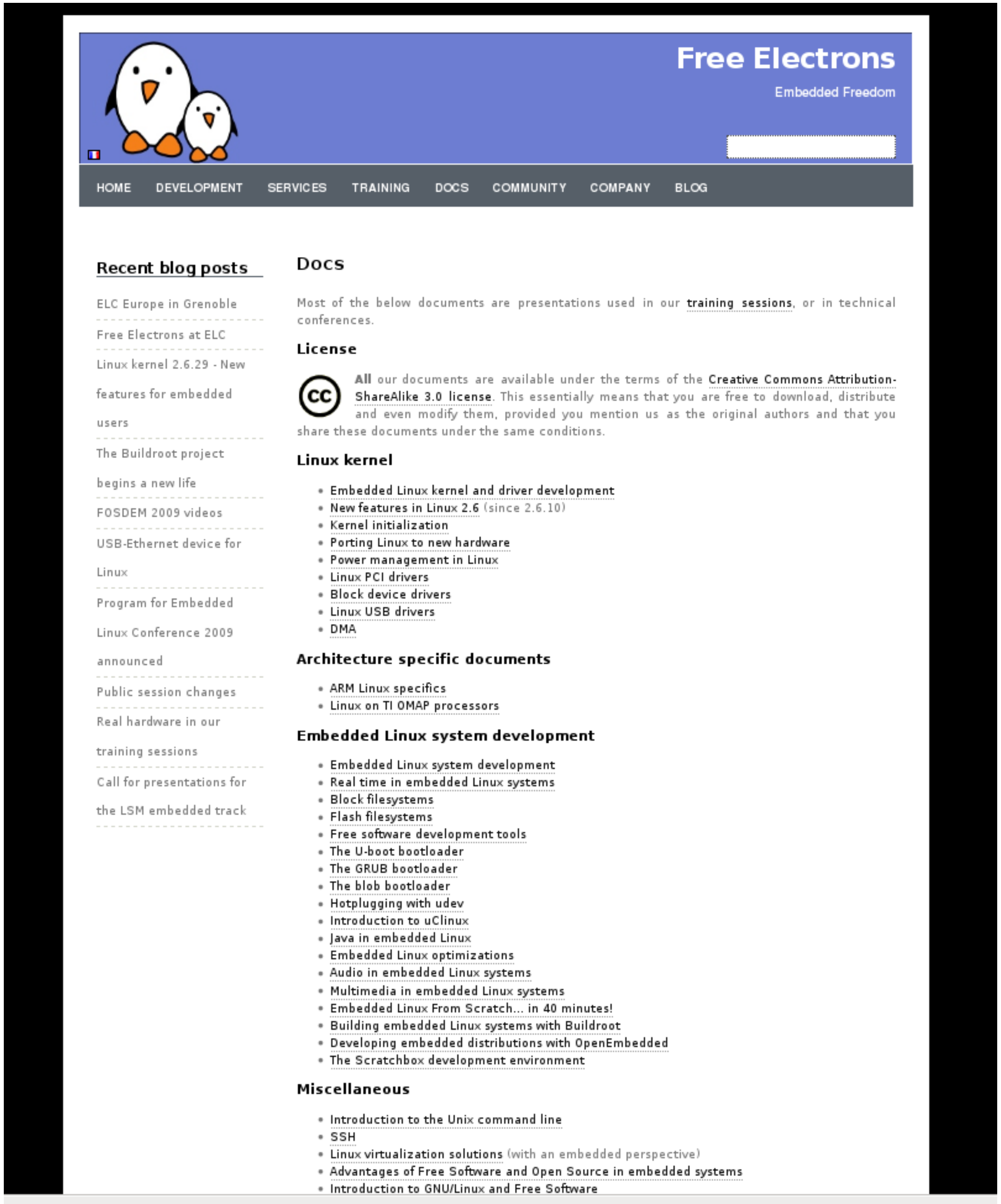
→	Speed increase	RAM reduction	Power reduction	Cost reduction
More speed			- CPU can run slower or stay longer in power saving mode	- Slower, cheaper CPU
Less RAM	- Faster allocations - Less swapping - Sometimes less cache flushing		- Fewer / smaller RAM chips: less dynamic and standby power. - CPU with less cache: less power	- Fewer / cheaper RAM chips - CPU with less cache: cheaper
Less space	- Faster application loading from storage and in RAM. - Sometimes, simpler, faster code.	- Less RAM usage	- Fewer / smaller storage chips: less power	- Fewer / cheaper storage
Less power				- Cheaper batteries - or cheaper AC/DC converter

The ultimate optimization tool!



- We have seen many ways to optimize an existing system.
- However, nothing replaces a good design!
- So, first carefully design and implement your system and applications with their requirements in mind.
- Then, use the optimization techniques to further improve your system and the parts that you reused (kernel and applications).

Related documents



Recent blog posts

- ELC Europe in Grenoble
- Free Electrons at ELC
- Linux kernel 2.6.29 - New features for embedded users
- The Buildroot project begins a new life
- FOSDEM 2009 videos
- USB-Ethernet device for Linux
- Program for Embedded Linux Conference 2009 announced
- Public session changes
- Real hardware in our training sessions
- Call for presentations for the LSM embedded track

Docs

Most of the below documents are presentations used in our training sessions, or in technical conferences.

License

All our documents are available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license. This essentially means that you are free to download, distribute and even modify them, provided you mention us as the original authors and that you share these documents under the same conditions.

Linux kernel

- Embedded Linux kernel and driver development
- New features in Linux 2.6 (since 2.6.10)
- Kernel initialization
- Porting Linux to new hardware
- Power management in Linux
- Linux PCI drivers
- Block device drivers
- Linux USB drivers
- DMA

Architecture specific documents

- ARM Linux specifics
- Linux on TI OMAP processors

Embedded Linux system development

- Embedded Linux system development
- Real time in embedded Linux systems
- Block filesystems
- Flash filesystems
- Free software development tools
- The U-boot bootloader
- The GRUB bootloader
- The blob bootloader
- Hotplugging with udev
- Introduction to uClinux
- Java in embedded Linux
- Embedded Linux optimizations
- Audio in embedded Linux systems
- Multimedia in embedded Linux systems
- Embedded Linux From Scratch... in 40 minutes!
- Building embedded Linux systems with Buildroot
- Developing embedded distributions with OpenEmbedded
- The Scratchbox development environment

Miscellaneous

- Introduction to the Unix command line
- SSH
- Linux virtualization solutions (with an embedded perspective)
- Advantages of Free Software and Open Source in embedded systems
- Introduction to GNU/Linux and Free Software

All our technical presentations
on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development

How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Free Electrons

Our services

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs