ELC 2009

System Size BOF

Michael Opdenacker
Bootlin

Rights to copy

Document sources:
https://bootlin.com/pub/conferences/2009/elc/

Document updates
https://bootlin.com/docs/

Corrections, suggestions,
contributions and translations are welcome!

Latest update: Jul 25, 2018

24 slides...

To avoid a tragic increase in the size of your system.

Why system size matters

- Because Linux wouldn't fit otherwise
- To leave more space for user data (media players)
- To keep things easier to maintain
- Lighter code is faster to boot
- We should stop size growth because we don't want to force people to use old kernels and old software.

Linux Tiny achievements

Merged features:

[x86] use ELF section to list CPU vendor specific code

[x86] configurable DMI scanning code

[mm] directly use kmalloc() and kfree() in init/initramfs.c

[x86] consolidate the definition of the force_mwait variable

inflate: refactor inflate malloc code

fs/buffer.c: uninline __remove_assoc_queue()

[x86] make movsl_mask definition non-CPU specific

[x86] move cmpxchg fallbacks to a generic place

[x86] configuration options to compile out x86 CPU support code

Configure out file locking features

Configure out AIO support

[PCI] allow quirks to be compiled out

[x86] remove PC speaker code

Work on multicast and ethtool configurability. Not merged yet.

Implemented
by Bootlin,
funded by CELF

Linux Tiny status

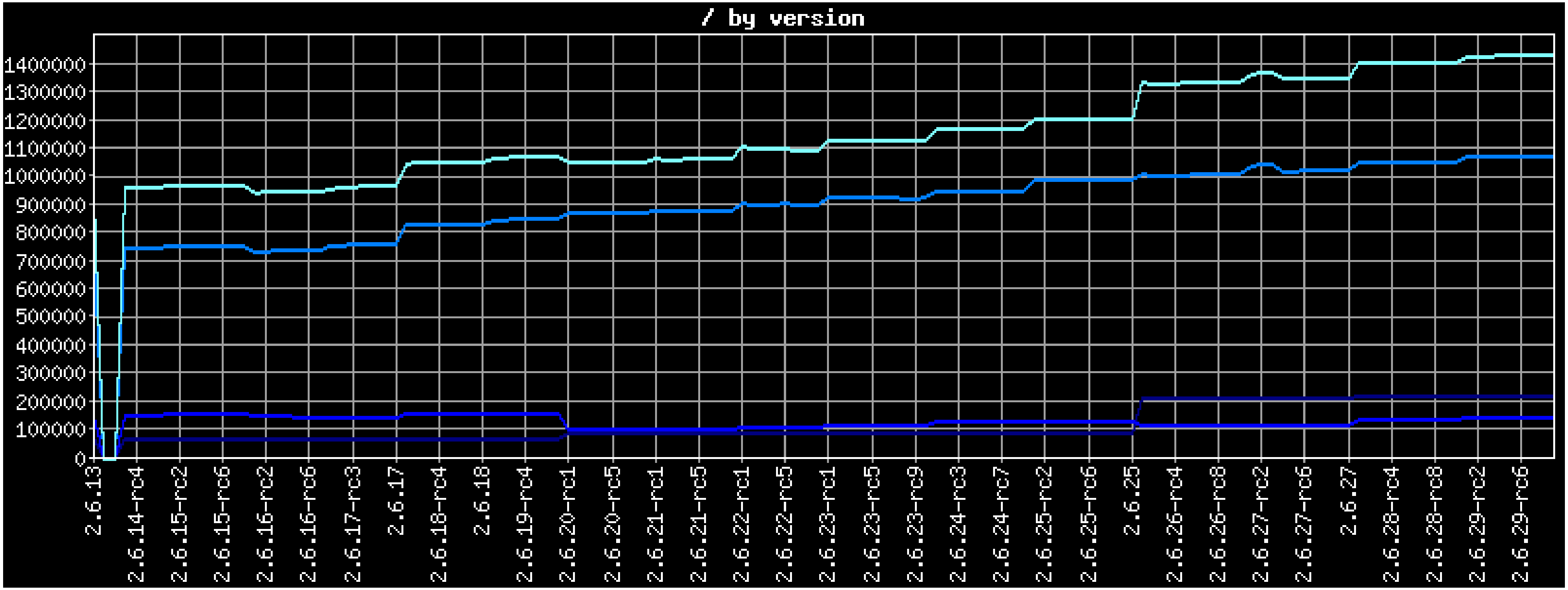
The diet must go on...

Stopped maintaining the patches
- Merge them or let them bitrot!

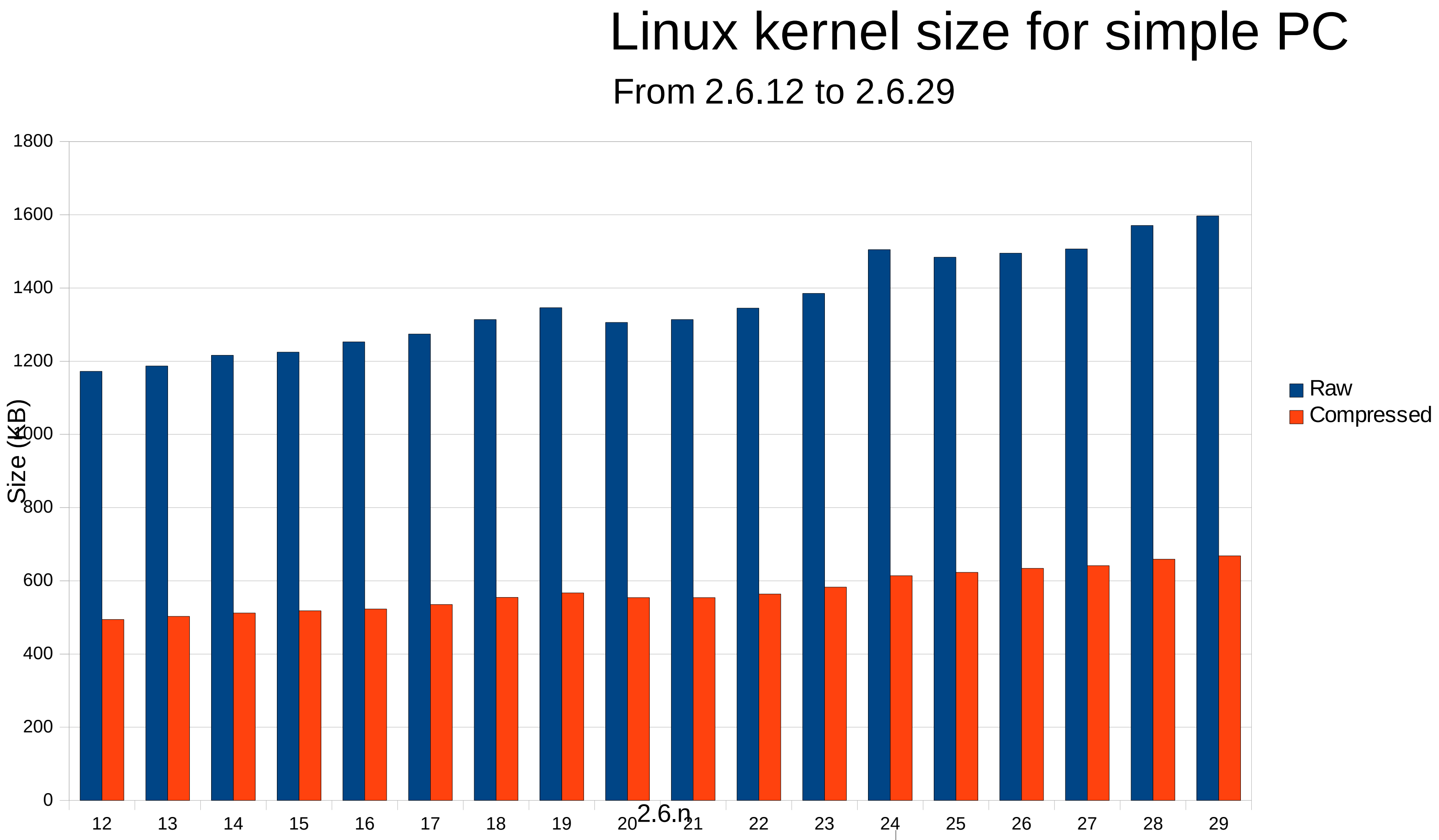
But the kernel continues to grow...
- Unavoidable progress of fate?

Bloatwatch report

http://www.selenic.com/bloatwatch/
Source code: http://www.selenic.com/repo/bloatwatch



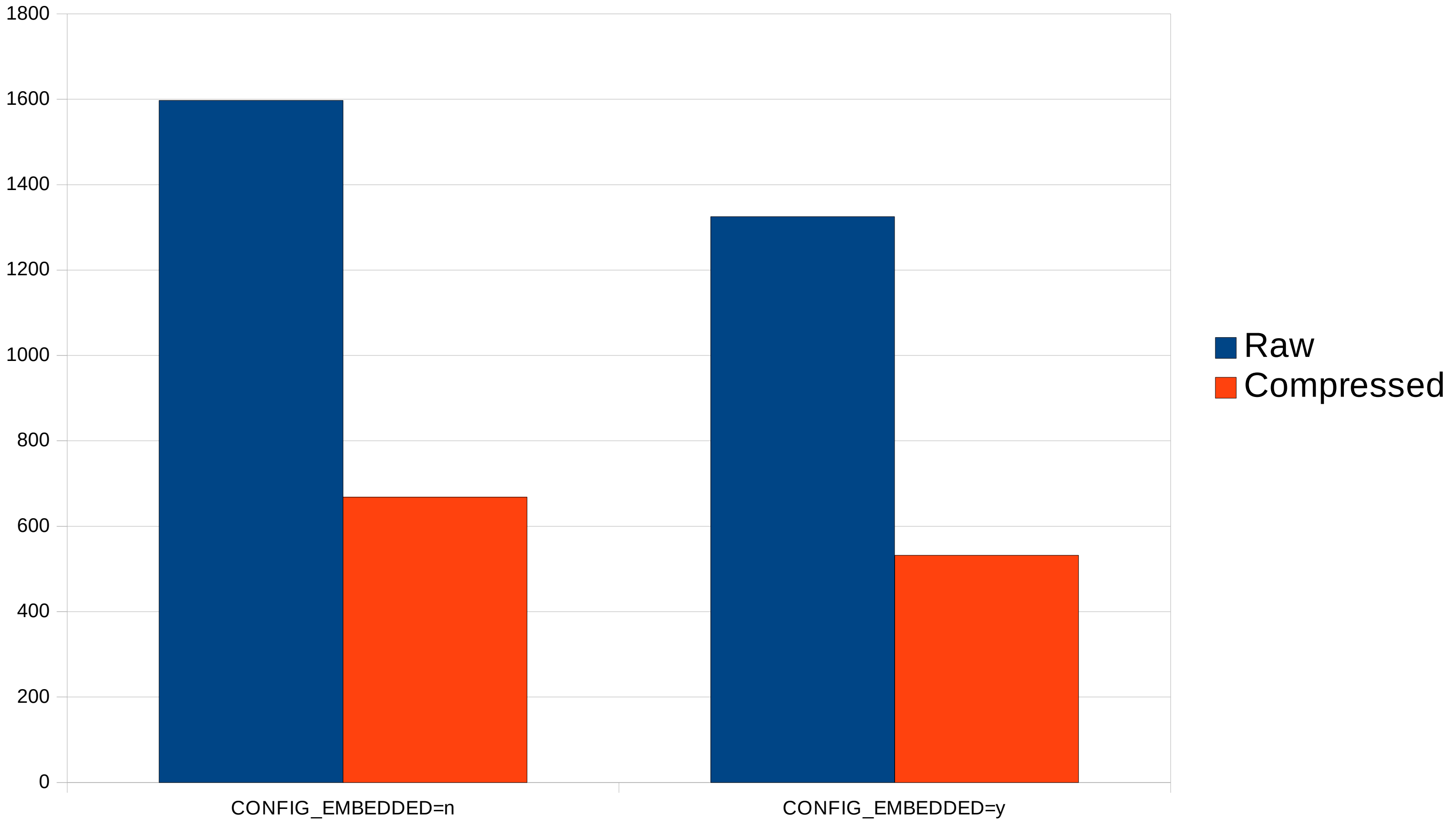
x86 kernel size example



► compiled with gcc 3.4!
(beware of compiler artifacts!)

With and without CONFIG_EMBEDDED

Same testcase. Also tested!



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

Linux Tiny ideas

Reduced /proc

Remove unused code when using a RAMFS:
readahead, swapping, pdflush (compiled unconditionally)

Just look at which files get compiled!
You will find things you probably don't need.

How to find kernel functions that are never executed?
What about using ftrace to find them (good idea to explore).

Move all debugging interfaces to debugfs

Your ideas?

Kernel size, really an issue?

- Still growing much slower than Moore's Law (which flash storage is supposed to follow).
- But perhaps still an issue for boot time:
  - A smaller kernel takes less time to copy to RAM
  - Keeping the kernel simpler also helps: less unused subsystems to initialize.

Compressed filesystems (1)

Great solutions to reduce system size,
available in the latest kernels:

UBIFS: compressed filesystem for flash (MTD) storage.
Like JFFS2, but without the poor performance.
Available since Linux 2.6.27.

SquashFS: lightning fast filesystem, perfect for all the parts of the root filesystem which can be kept read-only.
Available since Linux 2.6.29.

See my presentation on flash filesystems tomorrow (11:00 am, Imperial A)

Compressed filesystems (2)

Hey, what about block storage (USB flash drives, SSD)??

Read-only: use Squashfs

What solutions for read-write partitions?

- Does anyone use the FUSE based solutions?
- Any other suggestion?

Compiler switches

Standard -Os option.
Supported for compiling Linux

`-funit-at-a-time`
Made gcc do a much better job of inlining and dead code removal. No longer does anything according to gcc's manual.

-fwhole-program --combine
Equivalent to grouping all source files and making all variables static. Not longer offered in BusyBox options. What happened?

`-mregparm=3`
Seems to be x86 specific. Instructs the compiler to use registers for the first three function arguments.

See http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
for all available switches.

Use a lighter C library

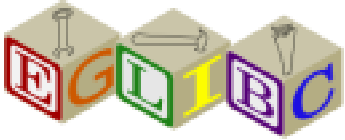
glibc: approximately 2 MB in size

uClibc good (usually < 500 KB).

- but often behind glibc in terms of features (floating point support, RT support...)

eglibc

eglibc



« Embedded glibc », under the LGPL
http://www.eglibc.org

Variant of the GNU C Library (GLIBC) designed to work well on embedded systems

Strives to be source and binary compatible with GLIBC

eglibc's goals include reduced footprint, configurable components, better support for cross-compilation and cross-testing.

Can be built without support for NIS, locales, IPv6, and many other features.

Supported by a consortium, with Freescale, MIPS, Montavista and Wind River as members.

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.

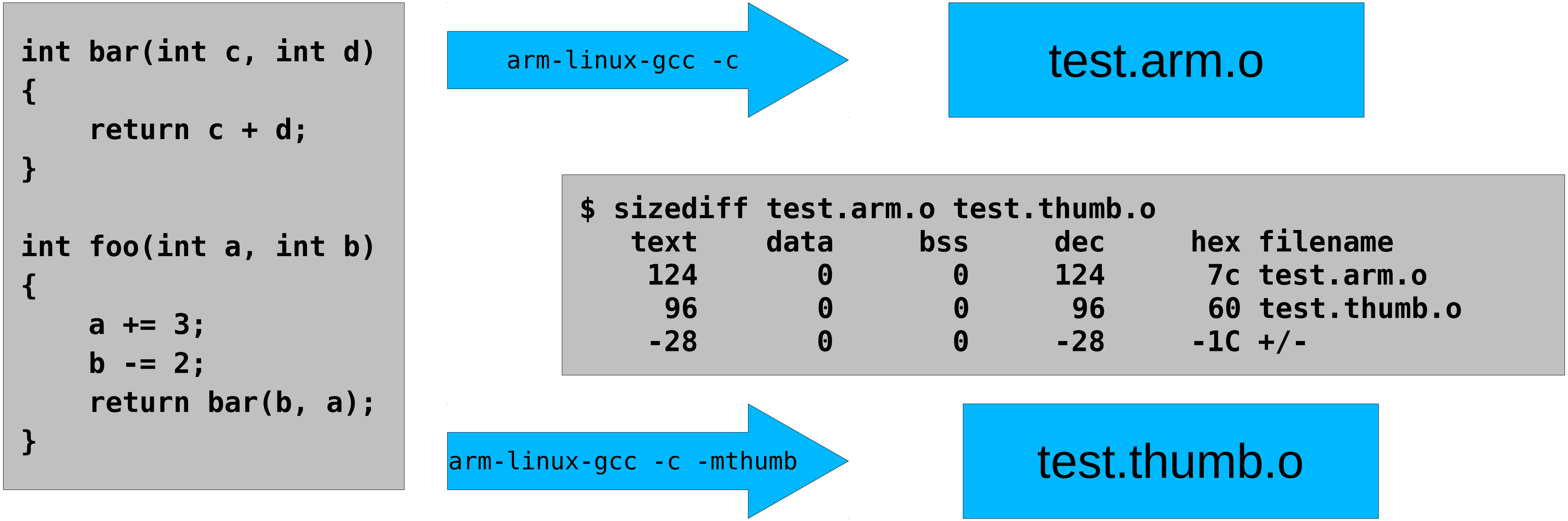
Can also take care of stripping executables and libraries.

However, requires to rebuild all the components from source. Would be nicer to achieve this only with ELF manipulations.

Anyone using it?

ARM Thumb (1)

Size gains on a small, non-representative example



```
int bar(int c, int d)
{
    return c + d;
}

int foo(int a, int b)
{
    a += 3;
    b -= 2;
    return bar(b, a);
}
```

```
$ sizediff test.arm.o test.thumb.o
   text    data     bss     dec     hex filename
    124       0       0     124      7c test.arm.o
     96       0       0      96      60 test.thumb.o
    -28       0       0     -28     -1C +/-
```

28 bytes reduction, 22% code size reduction

ARM Thumb (2)

Interworking: possible to mix ARM and Thumb code:
ARM for performance critical code
Thumb for code which size matters.

See http://bootlin.com/docs/arm-linux/ for details about how to generate and use Thumb code.

Anyone using this?

Thumb2: allows to get almost the same performance as ARM code, with almost the same size as Thumb.
No longer requires code switching.

Anyone having tried Thumb2?
Already supported by gcc since 2006.
Support for userspace Thumb2 included in Linux 2.6.26.

Questions

- Since you're here, size should be a concern to you
  - Why?
- What's biggest in your system?
- New techniques not listed here?

Resources

http://elinux.org/System_Size

http://bootlin.com/docs/optimizations/

```
rm -r *
  ;-)
```