There can be only one
The unified x86 architecture

Glauber Costa
glommer@redhat.com

Red Hat Inc.

October 9th, 2008

There Can Be Only One

Immortalizing the Linux Kernel

There Can Be Only One

Getting a head ahead

There Can Be Only One

McLeod is Busy so I did it

Roadmap

Brothers torn apart

- Linux had two ports for x86: i386 and x86_64

Brothers torn apart

- Linux had two ports for x86: i386 and x86_64
- i386 is creepy crappy, x86_64 not much better.

Brothers torn apart

- Linux had two ports for x86: i386 and x86_64
- i386 is creepy crappy, x86_64 not much better.
- Lots of code duplication

Tales of the pre-unification era

- Makefiles hack, like this: obj-o += ../../i386/kernel/myfile.c

Tales of the pre-unification era

- Makefiles hack, like this: obj-o += ../../i386/kernel/myfile.c
- Sharing happening under the hood.

Tales of the pre-unification era

- Makefiles hack, like this: obj-o += ../../i386/kernel/myfile.c
- Sharing happening under the hood.
- Bugs were raised, and in a lot of times, not noticed.

Tales of the pre-unification era

- Makefiles hack, like this: obj-o += ../../i386/kernel/myfile.c
- Sharing happening under the hood.
- Bugs were raised, and in a lot of times, not noticed.
- "Uhmm, lemme use this unsigned long in this arch/i386/kernel file, to represent a 32-bit quantity"

Flow is made difficult

- Bugs fixed in i386 would not always reach x86_64 and vice-versa

Flow is made difficult

- Bugs fixed in i386 would not always reach x86_64 and vice-versa
- Or they can be ported with errors.

Flow is made difficult

- Bugs fixed in i386 would not always reach x86_64 and vice-versa
- Or they can be ported with errors.
- Flow of code is prejudiced. It creates walls that shouldn't be there

What would you do if you had a wall like this?

Don't tell, let me guess...

The paravirt example

- `paravirt_ops`: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...

The paravirt example

- paravirt_ops: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...
- largely equal to i386!

The paravirt example

- `paravirt_ops`: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...
- largely equal to i386!
- cp arch/i386/kernel/paravirt.c arch/x86_64/kernel/paravirt.c. Works, but not very wise

The paravirt example

- paravirt_ops: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...
- largely equal to i386!
- cp arch/i386/kernel/paravirt.c arch/x86_64/kernel/paravirt.c. Works, but not very wise
- Code duplication and more important:

The paravirt example

- `paravirt_ops`: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...
- largely equal to i386!
- cp arch/i386/kernel/paravirt.c arch/x86_64/kernel/paravirt.c. Works, but not very wise
- Code duplication and more important: bugs fixed in a version, affecting both, may not get into the other.

The paravirt example

- `paravirt_ops`: x86_64 is different in a lot of ways, needs a lot of testing and a good PoC, but...
- largely equal to i386!
- cp arch/i386/kernel/paravirt.c arch/x86_64/kernel/paravirt.c. Works, but not very wise
- Code duplication and more important: bugs fixed in a version, affecting both, may not get into the other.
- Hey! Isn't it why we use generic constructs in the first place?

Roadmap

To arms!

To arms!

- Attempt 1: arch/i386, arch/x86_64

To arms!

- Attempt 1: arch/i386, arch/x86_64 and arch/x86

To arms!

- Attempt 1: arch/i386, arch/x86_64 and arch/x86
- arch/x86 gets the commons

To arms!

- Attempt 1: arch/i386, arch/x86_64 and arch/x86
- arch/x86 gets the commons
- If you touch a common file, you know you're doing it.

To arms!

- Attempt 1: arch/i386, arch/x86_64 and arch/x86
- arch/x86 gets the commons
- If you touch a common file, you know you're doing it.
- Changes your mindset

Troops march

- Works, but...

Troops march

- Works, but... not a full solution

Troops march

- Works, but... not a full solution
- Many files aren't equal, but could be.

Troops march

- Works, but... not a full solution
- Many files aren't equal, but could be.
- The more general the design, the better.

The merger

- if diff returns no output: move them to arch/x86

The merger

- if diff returns no output: move them to arch/x86
- Otherwise: arch/i386/kernel/foobar.c $\longrightarrow$ arch/x86/kernel/foobar_32.c

The merger

- if diff returns no output: move them to arch/x86
- Otherwise: arch/i386/kernel/foobar.c $\longrightarrow$ arch/x86/kernel/foobar_32.c
- Mechanical. No bugs expected. Works fine

The merger

- if diff returns no output: move them to arch/x86
- Otherwise: arch/i386/kernel/foobar.c → arch/x86/kernel/foobar_32.c
- Mechanical. No bugs expected. Works fine (Famous last words)
- Bisection

Roadmap

Soulmatch

- if the body doesn’t match, but the soul does:

Soulmatch

- if the body doesn't match, but the soul does:
- change the shape, but vmlinux should not deviate.

Soulmatch

```
   text    data     bss     dec     hex filename
4318765  569156  618348 5506269  5404dd vmlinux.old
4318765  569156  618348 5506269  5404dd vmlinux.new
```

Soulmatch

```
   text    data     bss     dec     hex filename
4318765  569156  618348 5506269  5404dd vmlinux.old
4318765  569156  618348 5506269  5404dd vmlinux.new

   text    data     bss     dec     hex filename
4318765  569156  618348 5506269  5404dd vmlinux.old
4318797  569156  618348 5506301  5404fd vmlinux.new2
^^^^^^^                 ^^^^^^^  ^^^^^^
```

Good integration vs Bad Integration

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club.

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head.

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

Let's not do it.

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

  Let's not do it.
- If there is *one* architecture, why bother?

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

Let's not do it.

- If there is *one* architecture, why bother?
- Tests on CONFIG_FEATURE are okay

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

Let's not do it.

- If there is *one* architecture, why bother?
- Tests on CONFIG_FEATURE are okay
- CONFIG_X86_IO_APIC: All x86_64 have one, but so what?

Good integration vs Bad Integration

- Tests on CONFIG_X86_XX kill baby seals.
- With a club. In the head. Very Hard.

Let's not do it.

- If there is *one* architecture, why bother?
- Tests on CONFIG_FEATURE are okay
- CONFIG_X86_IO_APIC: All x86_64 have one, but so what?
- Ok for temporary steps

Areas still needing attention

Search for CONFIG_X86_32,64: Usually denotes incomplete integration

file	occurrences
kernel/apic.c	31
kernel/io_apic.c	26
kernel/setup.c	20
kernel/cpu/common.c	20
kernel/ptrace.c	18
kernel/smpboot.c	16
kernel/cpu/amd.c	10
kernel/kprobes.c	9
kernel/i387.c	9
kernel/acpi/boot.c	9

Areas still needing attention

- About 100 files still have their _32 and _64 versions.
- Sometimes it's the right thing to do:

Areas still needing attention

- About 100 files still have their _32 and _64 versions.
- Sometimes it's the right thing to do: ex: page table code.

Roadmap

Analysis

- More robust x86 code:

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”

Analysis

- More robust x86 code: "This bug was there since RMS had no beard, and we never noticed"
- Feature richness:

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”
- Feature richness: This features existed for A and not for B. All of a sudden, it exists, and inherits years of testing

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”
- Feature richness: This features existed for A and not for B. All of a sudden, it exists, and inherits years of testing
- New features:

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”
- Feature richness: This features existed for A and not for B. All of a sudden, it exists, and inherits years of testing
- New features: I have to develop this kool-aid. Don’t have to port it to the other x86 variant

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”
- Feature richness: This features existed for A and not for B. All of a sudden, it exists, and inherits years of testing
- New features: I have to develop this kool-aid. Don’t have to port it to the other x86 variant
- Fewer Obvious bugs:

Analysis

- More robust x86 code: “This bug was there since RMS had no beard, and we never noticed”
- Feature richness: This features existed for A and not for B. All of a sudden, it exists, and inherits years of testing
- New features: I have to develop this kool-aid. Don’t have to port it to the other x86 variant
- Fewer Obvious bugs: I do know this code is used in a mixed word-size environment, with 2, 3 or 4 levels of page tables, etc

Analysis

- Most bugs are regressions.

Analysis

- Most bugs are regressions.
- Sometimes, code does get more complicated.

Thanks

- You all, for listening

Thanks

- You all, for listening
- People from Hamburg in general, for coming up with the Hamburger.