

Re: Use of C99 int types

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-04/1719.html>

From: Kyle Moffett (*mrmacman_g4_at_mac.com*)

Date: 04/06/05

Date: Wed, 6 Apr 2005 17:11:53 -0400

To: Renate Meijer <kleuske@xs4all.nl>

On Apr 06, 2005, at 07:41, Renate Meijer wrote:

> *On Apr 6, 2005, at 12:11 AM, Kyle Moffett wrote:*

>> *Please don't remove Linux-Kernel from the CC, I think this is an*

>> *important discussion.*

GAAH!!! Read my lips!!! Quit removing Linux-Kernel from the CC!!!

> *As I see it, there are a number of issues*

>

> *- Use of double underscores invades compiler namespace (except in*

> *those cases*

> *where kernel definitions end up as the basis for definitions in*

> */usr/include/*, i.e.*

> *those that actually are part of the C-implementation for Linux.*

It is these that I'm talking about. This is exactly my point (The

cases where

the kernel definitions are part of /usr/include).

> *- Some type that does not conflict with compiler namespace to replace*

> *the variety*

> *of definitions for e.g. 32-bit unsigned integers we have now.*

As I said, I don't care about this, so do whatever you want.

> *- Removal of anything prefixed with a double underscore from*

> *non-C-implementation*

> *files.*

ATM, much of the stuff in include/linux and include/asm-* is considered

"C-implementation" because it is used from userspace. If you want to

clean

that up and start moving abi files to include/kernel-abi or somesuch,

feel

free, but that's a lot of work

>> *Personally, I don't care what you feel like requiring for purely*

Linux-Kernel: Re: Use of C99 int types

>> *in-kernel interfaces, but __{s,u}{8,16,32,64} must stay to avoid
>> namespace collisions with glibc in the kernel include files as used
>> by userspace.*
>
> *Aye, but as I have pointed out several times, these types should be
> restricted
> to those files and *only* those files which eventually end up in the
> compilers
> includes. In every other place, they invite exactly the trouble they
> are intended
> to avoid.*

Precisely.

So if you want to make the millions of patches, go right ahead, be my guest. :-P
Until somebody steps forward to clean up the huge mess, nothing will get done.

> *So in every place except those files which may actually cause a
> namespace conflict or
> a bug because some newer version does not support __foobar, or changed
> the
> semantics. Since using any __foobar type implies relying on the
> compiler internals,
> which may change without prior notice, it is ipso facto undesirable.*

Except the kernel wants to be optimized and work and use what features are available.

The kernel uses __foobar stuff provided by the compiler because it has gccX.h files specifically designed to take compiler interfaces, provide backups when they don't exist, and use them (and their better checking) when they do.

>> *This is kinda arguing semantics, but:
>> A particular set of software (linux+libc+gcc), running in a particular
>> translation environment (userspace) under particular control options
>> (Signals, nice values, etc), that performs translation of programs for
>> (emulating missing instructions), and supports execution of functions
>> (syscalls) in, a particular execution environment (also userspace).*
>
> *Ok. And where exactly are linux and libc when compiling code for an
> Atmel ATmega32 (40 pin DIL) using gcc?*

Where do you get Atmel ATmega32 from? I only care about what symbols Linux can use, and as I've mentioned, when running under *Linux*, then it just so happens that *Linux* is part of my implementation, therefore the *Linux* sources, which by definition aren't used elsewhere, can assume they are part of

Linux–Kernel: Re: Use of C99 int types

said
implementation.

- > *The 'set of software' does*
- > **not* include any OS. Not Windows, not Linux, not MacOSX, since the*
- > *whole thing might be directed at a lowly microcontroller, which DOES*
- > *NOT HAVE ANY OPERATING SYSTEM WHATSOEVER.*
- >
- > *Nevertheless, gcc works fine.*

This is unrelated and off topic. Heck, you've even consented above that Linux can use

- >> *Without the kernel userspace wouldn't have anything, because anything*
- >> *syscall–related (which is basically everything) involves the kernel.*
- >
- > *Sure. The same goes for every other program. However, it would be*
- > *pretty*
- > *stupid to say the kernel is an integral part of (say) the Gimp . More*
- > *so, since*
- > *the Gimp and GCC run on completely different architectures aswell.*
- >
- > *By the same token, linux is part of XFree86 despite the fact XFree86*
- > *does not*
- > *require linux to run.*

But an XFree86 binary compiled on FreeBSD, or a GIMP binary compiled on FreeBSD, for the most part, will not run on Linux, because the compiler uses the `_Linux_` environment to build the binary, including the `_Linux_` headers and such. The built binary is nearly useless without Linux, but not vice–versa, hence even though the binary is not a derivative work of linux, it requires it to run.

- >> *Heck, the kernel and its ABI is `_more_` a part of the implementation*
- >> *than glibc is! I can write an assembly program that doesn't link to*
- >> *or use libc, but without using syscalls I can do nothing whatsoever.*
- >
- > *I can write entire applications using gcc without even thinking of*
- > *using*
- > *any 'syscall' or any other part of linux/bsd/whatever. Still... it's*
- > *gcc.*

Uhh, what exactly is your application going to do? So it wants to access memory, it faults to the kernel and gets stuff paged in. It wants to access a file, it does a syscall. If it wants to allocate more memory, it

Linux-Kernel: Re: Use of C99 int types

calls

into the kernel. This is all platform specific, and part of the implementation.

- > <Wishful Thinking>
- > *It would be nice if Linux became totally independent of any compiler,*
- > *or at least that*
- > *coupling between them would be minimal and that the amount of assembly*
- > *needed*
- > *would be minimal.*

If you feel like fixing the compiler to provide good enough interfaces, or fixing the kernel to abstract all of that out, then fine, but remember that the kernel has to deal *_directly_* with the hardware and is generally dependent on direct MMU twiddling, which *_can't_* be done from C :-P.

- > *It would be nice if linux defined and documented its own platform*
- > *specific types*
- > *somewhere in the arch directory, using a consistent (across platforms)*
- > *naming scheme*
- > *and used those types consistently throughout the kernel,*
- > *drivers, daemons and other*
- > *associated code.*

Got a patch?

- > </Wishfull Thinking>
- >
- > <Nightmare>
- > *Your scenario above. Never-ending streams of compatibility issues and*
- > *gcc drifting*
- > *further and further from the ISO-C standard and more and more*
- > *developers depending*
- > *on non-standard interfaces, linux growing ever more dependent on*
- > *support fro features*
- > *ABC and XYZ being implemented consistently cross platform, so that if*
- > *I want to use*
- > *gcc to compile for an AVR, i'm stuck with a shitload of linux issues,*
- > *kept "for backward*
- > *compatibility".*
- > </Nightmare>

Linux has a bunch of gcc headers that configure it based on the compiler version.

Change the compiler and we'll add another header, which, though messy, provides us some safety. It would be *_nice_* however, if the compiler had a gcc/types.h file

Re: Use of C99 int types

Linux-Kernel: Re: Use of C99 int types

that just provided it all for us, so we don't need to hardcode it all based on the architecture specified. Directly defined `__gcc_u32` types (Or whatever the GCC people like) would be even better.

```
>>> Nope. The syscall interface is employed by the library, no more,
>>> no less. The C standard does not include *any* platform specific
>>> stuff.
>>
>> Which is why it reserves __ for use by the implementation so it can
>> play wherever it wants.
>
> The C-implementation, which still does not include the kernel. At most
> a few header files, which are used as a basis for standard types by
> the C
> implementation, but no more. Any double underscore in a .c file is a
> blatant
> error. Most used in .h files are, too.
```

So how do I get `<linux/fb.h>` to work? There aren't "just a few", there are `__u32`, etc in `_everything_` with an `ioctl` or `syscall` interface, basically anything with an ABI.

```
> Fine. I assume it does. But #include <linux/fb.h> does not make the
> framebuffer (nor linux, for that matter) part of the c-implementation.
> From
> the two files mentioned above, only stdlib.h is.
```

Ok, so how do I fix `linux/fb.h` to `_not_` use `__u32`?

```
>> I want it to get the correct types, I don't want it to clash with or
>> require the
>> libc types (My old sources might redefine some stdint.h names, and I
>> don't want it
>> to clash with my user-defined types.
>
> Redefining stdint types is (for this reason) a Bad Idea.
```

So how do I use them in `<linux/*.h>`?

```
>>> Anything you like. 'kernel_' or simply 'k_' would be appropriate.
>>> As long as you do not invade compiler namespace. It is separated
>>> and uglified for a purpose.
>>
>> But the _entire_ non _ namespace is reserved for anything user
>> programs want to do with it.
>
> The above prefix was an alternative to using a double underscore
> prefix. Using *no*
```

Linux-Kernel: Re: Use of C99 int types

- > *prefix, should not conflict with the compiler, excepting, of course,*
- > *the types required by*
- > *the standard.*

But these are also used by c programs.

- >> *When a program*
- >> *compiled as ppc32 gets run on my ppc64 box, the kernel understands*
- >> *that anything pushed onto the stack as arguments is 32-bit, and must*
- >> *use specifically sized types to handle that properly.*
- >
- > *And thus you end up using a 32-bit interface between a 64 bit OS and a*
- > *64 bit*
- > *application? Or two separate syscall interfaces?*

What about "When a program compiled as *ppc32*..." don't you get? I have my ppc32 program. It doesn't support the new ppc64 syscall or ioctl interface, because it's 32-bit. I didn't say anything about ppc64 programs, which use the new ppc64 syscall interface.

- > *Neither option seems very desirable. What about pointers which are*
- > *32 bit on one platform and 64 on the other? IOW, i'm not sure*
- > *"backwards*
- > *compatibility" is the thing to strive for. We all know what it did to*
- > *Intel-processors*
- > *and if it means having to jam data from a 64-bit App to a 64 bit OS*
- > *through a 32-bit*
- > *syscall interface, it stinks.*
- >
- > *Especially since most packages need only to be recompiled for the new*
- > *situation and*
- > *source (commonly) is available.*

But what about when I boot between ppc32 and ppc64 on my G5, because PPC64 doesn't support the driver for some piece of hardware? Why can't I just use all my old 32-bit binaries? The G5 has full 32-bit compatibility, why shouldn't the kernel?

Cheers,
Kyle Moffett

-----BEGIN GEEK CODE BLOCK-----

Version: 3.12

GCM/CS/IT/U d- s++: a18 C++++>\$ UB/L/X/*++++(+)>\$ P+++(++++)>\$
L++++(++++) E W++(+) N+++((+) o? K? w---- O? M++ V? PS+() PE+(-) Y+
PGP+++ t+(++) 5 X R? tv-(--) b++++((+) DI+ D+ G e->++++\$ h!*(())>+++ \$ r

Re: Use of C99 int types

Linux-Kernel: Re: Use of C99 int types

!y?(-)

-----END GEEK CODE BLOCK-----

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>