

Re: [RFC/PATCH 0/22] W1: sysfs, lifetime and other fixes

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-04/6156.html>

From: Evgeniy Polyakov (johnpol_at_2ka.mipt.ru)

Date: 04/25/05

Date: Mon, 25 Apr 2005 23:26:58 +0400

To: dtor_core@ameritech.net

On Mon, 25 Apr 2005 11:32:14 -0500

Dmitry Torokhov <dmitry.torokhov@gmail.com> wrote:

> On 4/25/05, Evgeniy Polyakov <johnpol@2ka.mipt.ru> wrote:

> > On Thu, 2005-04-21 at 09:31 -0500, Dmitry Torokhov wrote:

> > >

> > > OK, that is what I am aying. But why do you need that attribute with

> > > variable name and a bin attribute that is not really bin but just a

> > > dump for all kind of data (looks like debug one).

> >

> > bin attribute was created for *lm_sensors* scripts format – it only caches

> > read value.

> > I think there might be only 2 "must have" methods – read and write.

> > I plan to implement them using connector, so probably they will go away

> > completely.

> >

> > ...

> > > You will not be able to cram all 1-wire devices into unified

> > > interface. You will need to build classes on top of it and you might

> > > use connector (I am not sure) bit not on w1 bus level.

> > > ...

> >

> > connector allows to have different objects inside one netlink group,

> > so it will use it in that way.

> > I think only two w1 methods must exist – read and write,

> > and they must follow protocol, defined in family driver.

> >

> > No, I think there should not be any "must have" methods on w1_bus

> > level. What you really need (and this needs to be coordinated with

> > other sensors people) is a "sensors" class hierarchy that will define

> > classes like "temperature sensor", "fan", "vid", etc. Then your w1

> > family drivers, when bound to a slave, will create needed class

> > devices. i2c drivers will do the same, and your superio, and I'll be

> > able to change i8k driver just for kicks. Then your usespace would not

> > care what _bus_ a particular sensor is sittign on and will be

> > presented with a unified interface. Look at your NIC example –

Linux–Kernel: Re: [RFC/PATCH 0/22] W1: sysfs, lifetime and other fixes

- > *userspace does not care if NIC is sitting on a PCI, ISA, PCMCIA or USB*
- > *bus – it's all the same. And your classes can use netlink as a*
- > *transport mechanism – fine, why shouldn't they... But it will be*
- > *available for entire kernel, not only w1 bus..*
- >
- > *Once again, bus code is not the right level to define interface with*
- > *userspace. There just way too many different devices acn be connected*
- > *to the same bus. You need to separate them into classes andd efine*
- > *interface tfor a class. And class does not have to be confined to a*
- > *signe bus, it can span across several buses, providing unified*
- > *interface to a group of similar objects.*

Heh, that would be nice, but that requires a *_lot_* of changes, so it is only theory, at least for now.

Reality is that w1 devices must be managed from userspace, it can be implemented using 2 system calls – read and write, while they are implemented using sysfs and thus device/driver model, it requires different locking, which may race – and I will fix it. Other implementation could be connector's calls – they can be used in i2c core and *_are_* used in superio core, it will put it into w1 if it will be committed in. Probably implementation will be similar to superio one...

- > ...
- >> *If I understand you correctly, "lifetime rules" are implemented in a*
- >> *following way:*
- >> *when object is created it has 0 refcnt, each access increments it and*
- >> *must*
- >> *decrements when access is finished.*
- >
- > *No, not each access (well, depending on what you mean by access).*
- > *Normally, when you create an object you set it's refcount to 1 (becase*
- > *there is 1 owner – you). Evry time you pass that object to another*
- > *thread of execution (process) you need to increment reference count*
- > *and every time you donr with using object you decrement reference*
- > *count. Last user needs to destroy the object. See*
- > *Documentation/kref.txt*

The behaviour exactly the same, but implementation always waits on remove when others finish. No problem here.

- >>>>
- >>>>> *w1–bus–ops.patch*
- >>>>> *Cleanup bus operations code:*
- >>>>> *– have bus operatiions accept w1_master instead of unsigned long and*
- >>>>> *drop data field from w1_bus_master so the structure can be statically*
- >>>>> *allocated by driver implementing it;*
- >>>>> *– rename w1_bus_master to w1_bus_ops to avoid confusion with w1_master;*
- >>>>> *– separate master registering and allocation so drivers can setup proper*
- >>>>> *link between private data and master and set useable master's name.*

> > > >
> > > > *I strongly object against such changes.*
> > > > *1. w1 was designed in the way that w1 bus master drivers do not*
> > > > *know about other w1 world. It is very simple and very low-level*
> > > > *abstraction,*
> > > > *that only understands how to do low-level functions. It is not needed*
> > > > *do know about w1_master structure and even about it's existence.*
> > >
> > > *Well, it does need to know about w1_bus_master structure, which is*
> > > *pretty much the same. And it allows having static bus_ops allocation*
> > > *and removes need for casting from unsigned longs...*
> >
> > *w1_bus_master structure is low-level physical operations.*
> > *It completely does not know about logical links to other*
> > *w1 core objects, Why _bus_ driver should know about logical*
> > *objects on top of it? That is why they are separate.*
> > *Even if they are not too different (and actually they _are_ different)*
> > *from your point of view, they differ in abstraction model.*
> > *Bus master driver – is like NIC driver, it does not know about the rest*
> > *of*
> > *the network stack, but you want it to have all info about neighbours,*
> > *routes*
> > *and so on...*
> >
>
> *Ok, look at the drivers implementiog NICs – struct netdevice. The*
> *define open() and close() methods, but they also know a little bit*
> *about netdevice, like how to get rivate data (netdev_priv) and some*
> *other stuff. That is exactly what I have done WRT w1_master and*
> *w1_bus_master. Again, this allows to have w1_bus_master (w1_bus_ops)*
> *statically allocated and not piggy-back it to w1_master memory*
> *allocation. And we not have better type-safety because we don't pass*
> *unsigned longs and up/down cast them everywhere. And you don't need to*
> *"search" for a master device using "data" as a cookie. If you want we*
> *can have w1_master_priv to access w1_master->priv instead of*
> *referencing it directly.*

Neither network driver knows about how skb are used.

NIC's device driver does not know about neighbours, routes, filters and so on.

It only moves data to the physical layer, how data is managed over it is not in driver's competence.

w1 bus master driver only knows how to move data to the wire, it does not know about how that data was moved there, from/to which slave it is originated and so on.

Bus master driver is low-level part that lives in it's own driver, while w1 core only knows about higher-layer w1_master objects.

Like network stack operates on device driver (in our case it is w1 bus master driver) through dev->something(), and we have here dev->bus_master->something(), network core operates over routes using dst/rt – w1 has dev->slist and so on.

You may say that why call through dev->bus_master->something()
when we may call dev->something(), I can say that
w1_master itself is like a stack in network -
it knows about it's routes (slave devices),
it knows about it's low-level driver (bus master),
it has proper locking (xmit lock in network).

Bus master driver is absolutely separate object
from w1_master structure and logical object itself.

> > > > 3. You broke netlink allocation routing - it may fail and it is not
> > > > fatal.
> > >
> > > Because it is going away in later patches ;)
> >
> > This is wrong - netlink notification is used and will be moved to
> > connector
> > interface later.
>
> But not at the w1_bus level, please.

How do you suppose to notify about alarm condition?
Not from bus layer?

Who does send "link is down" messages? It is not the same
as device is present and found, it like "w1 device has something to read".
For example w1 ds18s20 thermal sensor may send information
about "85 degree problem" - it is read when sensor did not
finished temperature transformation yet, how non-bus layer may know about it?

Your idea about classes over the various buses is good,
but unfortunately it is utopia, at least for now,
so let's create w1 core layer (which, btw, is not only bus,
which is managed by bus master driver, but also some logic over it,
one may call it w1 stack, stack can send such a messages, doesn't it)?

> > > >
> > > > w1-drop-netlink.patch
> > > > Drop custom-made hotplug over netlink notification from w1 core.
> > > > Standard hotplug mechanism should work just fine (patch will follow).
> > > >
> > > > netlink notification was not created for hotplug.
> > > > Also I'm against w1 hotplug support, since hotplug is quite rarely used
> > > > in embedded platforms where the majority of w1 devices live.
> > >
> > > kobject_uevent does notification over netlink so I do not understand
> > > why custom approach is better. You don't really need to use script.
> >
> > kobject is too big for that. It is used exactly for kobject changes.
> > Custom netlink notifications are created for w1 specific objects
> > and it's control. You can not control w1 slaves/masters using hotplug.
>

- > *Hotplug is a unified mechanism for notifying userspace of new devices.*
- > *Not only on w1 bus but everywhere. Stop inventing solutions useable*
- > *for w1 bus only. And if I for some reason don't want to use netlink –*
- > *well I can with gebneric hotplug solution but not with your.*

The problem is that there is not only exist/not events, that may be sent, as I said in previous mail.

You suggest to limit it in that way – this is wrong.

Feel free to `_add_` hotplug support, but not replace notification with it.

- >>>> *w1-drop-control-thread.patch*
- >>>> *Drop control thread from w1 core, whatever it does can also be done in*
- >>>> *the context of w1_remove_master_device. Also, pin the module when*
- >>>> *registering new master device to make sure that w1 core is not unloaded*
- >>>> *until last device is gone. This simplifies logic a lot.*
- >>>>
- >>>> *Why do you think master can be removed in safe context only?*
- >>>>
- >>> *Can you show me example where you remove master from an interrupt*
- >>> *context or a tasklet? I doubt you will ever see one.*
- >>
- >> *As I said I have feature requests for ability to export w1 devices*
- >> *outside w1 core.*
- >> *Probably it is due to it's private non-GPL usage, so it is not created,*
- >> *but it is usefull feature actually and we can not know what will*
- >> *happen in what context when we export master/slave devices.*
- >
- > *Look at your present w1_remove_master_device. It sleeps. Sop there is*
- > *no need for a separate thread, callers must be able to sleep anyway.*

That is exactly why control thread exists – to manage sleepable operations!

- >> *w1 slaves can be found on the bus without search method reaction*
- >> *implemented in it's asic, btw.*
- >> *And it is _very_ usefull to add/remove slaves using external command but*
- >> *not using*
- >> *automatic detection in search methods.*
- >
- > *But the request for that will come from userspace with is perfectly*
- > *able to sleep. You are over-engineering and making kernel code*
- > *unnecessarily complex without thinking it through.*

Connector's requests come from BH context.

Only module unloading comes with good context (in our case we do not get read/write operations),

but I do not want to limit the system only for that kinds of events.

- >>>> *I have feature requests for both adding/removing and exporting*
- >>>> *master devices and slaves to the external world.*
- >>>>
- >>>> *External as in userspace? It (user thread) can wait just fine...*

- > >
- > > *Exporting them into other kernel modules.*
- > > *We do not know in what context that structures will be used there.*
- >
- > *Why other kernel modules would be interested in raw access w1_slaves?*
- > *C are to give an example?*

Concider w1 battery slave device, which exports unified interface to the userspace.

It requires access that can be obtained from different generic module [like existing kernelspace/userspace protocol, proper device files and so on],

which will implement only read/write interface.

It's read method will get_slave_by_something() and read it's data or do something else, then generic module will use that data.

Generic buttry monitor will not scan w1 bus for it's devices, since it even does not know about w1, it only understands reading/writing operations.

Placing operations needed for that module into w1_bat.c and hope that noone will implement new battery monitor subsystem or will adopt all battery users to use only that nterface is naive.

There is at least possibility to create such a model with existing design [and it is very easy], but your changes broke it, although it could be changed...

- > > > *Control thread is also the place in which we kick all devices*
- > > > *when we need it, but not only when we need to remove w1 core module.*
- > > >
- > > > *Define kicking for me please...*
- > >
- > > *Removing master device using netlink command for exaple.*
- >
- > *Wrong level. You need to start with device implementing w1_bus_master*
- > *(w1_bus_ops) to remova dangling data structures). Easiest way I think*
- > *it have the driver compiled as a module and remove it altogether – why*
- > *keep it if you don't need master?*

1. master can be removed by command. It is not in process' context. Current thread can remove only all object at once, but nevertheless I do not want to limit it.

2. control thread can add/remove new slaves by request. Usefullness of that ability was pointed in my previous e-mail, but you skipped that part.

- > > > > *w1-master-attr-cleanup.patch*
- > > > > *Clean-up master attribute implementation:*
- > > > > *– drop unnecessary "w1_master" prefix from attribute names;*
- > > > > *– do not acquire master->mutex when accessing attributes;*

>>>> – move attribute code "closer" to the rest of master code.
>>>>
>>>> *Ok, but slave count and slaves attributes itself requires that mutex.*
>>>
>>> *They are gone. You can scan sysfs to get your slaves and count. Kernel*
>>> *does not need to do that.*
>>
>> *I created that files exactly for reaason to not scan the tree, but only*
>> *read one [two] files :)*
>
> *The less code in kernel that produces data availavle elsewhere the better :)*

Does 3 lines of code for reading slave's names is too big
price for not scanning the whole /sys/bus/w1/w1_master1/ directory?

>>>> *w1-master-cleanup.patch*
>>>> *Clean-up master device implementation:*
>>>> *– get rid of separate refcount, rely on driver model to enforce*
>>>> *lifetime rules;*
>>>> *– use atomic to generate unique master IDs;*
>>>> *– drop unused fields.*
>>>>
>>>> *That patch is very broken.*
>>>> *I completely against it:*
>>>> *1. it breaks process logic – searching can be interrupted and stopped,*
>>>> *thread will exit on signals.*
>>>
>>> *Interrupted/stopped from userspace?*
>>
>> *Your loop waits only until interrupt happens – it can be delivered from*
>> *anywhere.*
>
> *No, only root can kill kernel therad so it is pretty safe. And hey, if*
> *a thread goes mad maybe it's a good thing that it can be killed.*

And if it exits – it breaks the logic – user can not know the state
of the master device when thread is exited, but module was not removed
by request.

>>>> *w1-family-is-driver.patch*
>>>> *Convert family into proper device-model drivers:*
>>>> *– embed driver structure into w1_family and register with the*
>>>> *driver core;*
>>>> *– do not try to manually bind slaves to familes, leave it to*
>>>> *the driver core;*
>>>> *– fold w1_family.c into w1.c*
>>>>
>>>> *Why do you break it?*
>>>> *They were separated intentionally – it simplifies code review,*
>>>> *it is completely different logical models, family processing*
>>>> *is not hte same as slave.*

> > >
> > > *Masters, slaves and families are all objects of W1 kernel bus. With
> > > cutting a bunch of fat from family code it does not make sense to keep
> > > them separate anymore.*
> >
> > *What you do is exactly the same that already exist, but using other
> > model.*
> > *No need to dig into device model in a such way.*
> ...
> > *the problem is that device model[which is not the main part of the w1
> > system]
> > is interfere to the existing locking schema [which is quite big and
> > allows very flexible object manipulation], and you suggest to almost
> > completely
> > replace one with another.*
>
> *device model is here to _use_ it. It already implements bunch of stuff
> you have to re–implement if you do it "your own way" and it is already
> debugged much better than your solution. And if there is a problem
> with driver core implementation – well, more people are looking at it
> and are more likely to discover a problem and offer a solution.*
>
> *I do not understand why you are against full integration with device
> model – it does simplifies and unifies the code.*

Because it is not needed here – and even if it could be integrated more closer – your changes broke too many special design cases, which are not acceptable.

> > *With such changes how to increment slave's usage counter? module_get
> > (w1_family)?*
>
> *Actually if you need it it would be get_device(&w1_slave->dev). And if
> you need pin family object you would get
> get_driver(&w1_family->driver). But I don't think you will needed it.
> Actually, at some point I had w1_family_get() implemented as a wrapper
> to get_driver() but since it does not seem to be needed I dropped it.*

We can not do it in that way.

1. low–level bus master code differs from what w1_master is.
2. family itself is different from slave object.

Having that we need to create w1_master/w1_family device/driver model locking + w1_bus_master/w1_slave locking or move them to device/driver model too. Why it is needed I still do not see, there is proper locking schema, which is broken in the places where existing model touches device/driver one, and it will be fixed.

> > > > *and it will not be possible if one just blindly gets/puts module's
> > > > refcnt.*
> > >

> > > *Only wire.ko is pinned. You are still free to remove family drivers or
> > > master drivers (or killing their objects somehow). It is only core
> > > that is pinned to make sure that release functions are available when
> > > object finally goes away.*
> >
> > *If we remove slave device, we must be sure it's object is freed
> > when appropriate kobject is released.*
> > *The same is with family itself.*
>
> *Right.*
>
> > *No, we need either replace all locking with device driver model,
> > or properly operate with existing one.*
>
> *And it was done. Well, not locking, but pinning of the objects. It was
> all moved to device model. It may not be visible ;) but it is there.*
> *Locking is still there as well.*

We need here either `_only_` device/driver locking (locking here and above is not only locking itself, but usage counters and all corresponding events too), or mix of the existing schema and device/driver model.

The first approach already has too many issues, which probably can be resolved, but probably don't, existing mixing model works except one case, which I will fix.

It will use device/driver model and it's remove callback to free resources.

Now I have following agenda for w1:

1. wait until existing changes are committed
2. put your's and Adrian's cleanups
3. fix w1 sysfs usage
4. commit 2 and 3
5. implement hotplug using your patch, but not instead of existing notification.
6. commit hotplug changes

I strongly believe that after [3] there will be no moot points.

Thank you.

> --
> *Dmitry*

Evgeniy Polyakov

Only failure makes us experts. -- Theo de Raadt

-

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/>