## Subject: Re: [PATCH] usbatm: Update to use the kthread api. Posted by Duncan Sands on Fri, 15 Dec 2006 09:42:37 GMT

View Forum Message <> Reply to Message

```
Hi Eric,
```

```
>>> The paradigm in a kthread world for waking up kernel threads is by
> >> calling kthread_stop, and then for testing if a kernel thread should
>>> stop is by calling kthread should stop.
> >
> > I considered this, but rejected it because of this comment:
>> * kthread_stop - stop a thread created by kthread_create().
>> * ... Your threadfn() must not call do_exit()
>> * itself if you use this function! ...
> > and this one:
>> * ... @threadfn can either call do exit() directly if it is a
>> * standalone thread for which noone will call kthread stop(), or
>> * return when 'kthread_should_stop()' is true (which means
>> * kthread stop() has been called).
> >
>> Most of the time the kernel thread starts, performs heavy_init,
>> and exits. The above comments seem to imply that it is wrong
> > to call do_exit if kthread_stop might be called, and wrong to
> > return if kthread_stop has not been called. This seems to exclude
>> the case where kthread stop is sometimes, but not always, called,
>> and the thread sometimes exits without kthread stop having been
> > called. But perhaps I misunderstood, since it seems there is kthread
> > code to handle the case of a threadfn that returns without kthread stop
> > having been called, witness this comment:
        /* It might have exited on its own, w/o kthread_stop. Check. */
>> It's still not clear to me, so if you can enlighten me, please do!
>
> This is a good point. I was going to say we could work around
> this with checks in usbatm_do_heavy_init but that appears racy.
```

presumably the problem is that if the thread has spontaneously exited, and afterwards disconnect calls kthread\_stop, then things go boom. The same problem exists (though with lesser consequences) when sending a signal. There is already code in usbatm to avoid this problem with signals. Why not just recycle it in the kthread\_stop case? I guess there is no problem if you can guarantee that the following occurs: if kthread\_stop is ever called for the kthread, then the kthread only exits after seeing kthread\_should\_stop return true.

> I guess I need to think a little more. I remember seeing this

> and not worry about it because SIGTERM didn't seem to > be caught, so it didn't appear needed. >>> I think the reduction in complexity and the increase in uniformity >>> is most likely worth it. > >> >>> If all else fails I'm happy with something simpler like Cedric's >>> patch which takes care of the things that I currently have a problem >>> with, but I'm willing to work through this to make it a through > >> cleanup. > > >> You have a problem with the pid, right? Well, that is easily >> cured in itself. I'll spin a patch for it a bit later, unless >> someone else gets there first. And if you can confirm that kthread\_stop >> can be used in this situation (i.e. thread can spontaneously return > > without kthread\_stop) then I'm happy to convert everyone over to checking > > kthread should stop. > To be clear I have a problem with using numeric pids of kernel threads, Yes, this is a problem with usbatm at the moment. > and with spawning threads from a possibly user space environment. Not the case with usbatm. It is always spawned from khubd. > So on my hitlist are (kill\_proc, daemonize, and kernel\_thread). > If it the original process is a user space thread it is possible to > capture pieces of the user space environment unintentionally daemonize > is supposed to fix that but only does for the pieces of user > space environment that people have anticipated you can capture. Best wishes, Duncan. Containers mailing list

https://lists.osdl.org/mailman/listinfo/containers

Containers@lists.osdl.org