Subject: Re: [PATCH] usbatm: Update to use the kthread api.
Posted by Duncan Sands on Thu, 14 Dec 2006 10:44:10 GMT
View Forum Message <> Reply to Message

Hi Eric, thanks for looking into this.

> During driver initialization if the driver has an expensive
> initialization routine usbatm starts a separate kernel thread for it.
>
> In the driver cleanup routine the code waits to ensure the
> initialization routine has finished.
>
> Switching to the kthread api allowed some of the thread management
> code to be removed.
>
> In addition the kill_proc(SIGTERM, ...) in usbatm_usb_disconnect was
> removed because it was absolutely pointless. The kernel thread did
> not handle SIGTERM or any pending signals, so despite marking the
> signal as pending it would never have been handled.

This is wrong, the signal is used. Let me explain the context, then why signals are important. USB ATM modem drivers register themselves with the usbatm core, which organizes the interaction between the USB layer, the ATM layer and the modem driver. Some modems require initialization that cannot be performed in the USB probe method. When I say "cannot" here, you need to understand that this is mainly about quality of service, though there are some correctness issues: initializing these modems takes typically 5 seconds or more. If the initialization was done in probe, all other USB device initialization/ disconnection would have to wait for it to finish (USB probe/disconnect is globally serialized, being run from the khubd kernel thread). This is unacceptable, so usbatm provides an easy way to have the extra initialization run from within it's own kernel thread: the modem driver registers a heavy_init method with usbatm; at the end of probe, heavy_init is run in its own kernel thread. In fact, I've been asked by Alan Stern to generalize this functionality into the USB core itself, since something like this is needed by a pile of USB drivers.

An important consideration: what if heavy_init is still running when the modem is disconnected? The disconnect method cannot exit until the kernel thread has exited; horrible mayhem could result otherwise. Thus disconnect has to wait for the kernel thread to finish. That means that the whole USB subsystem has to wait for the kernel thread to exit. This is problematic, from a quality of service point of view, if heavy_init takes a long time to finish. For example, the following line is executed by the heavy_init in speedtch.c:

msleep_interruptible(1000);

This is relatively mild, but already shows the problem: disconnect can take more than one second to exit. There are much worse cases (more on this later).

In short, the usbatm core needs a way to tell the heavy_init method that the game is up (due to disconnect). I chose to have it send a signal to the kernel thread. This seemed to be the simplest way. If the sending of a signal is removed, something else will have to replace it. Before I discuss how the signal is handled by existing heavy_init methods, I should point out that even if none of the existing heavy_init methods made any use of the signal, it would still be wrong to remove it: a not-yet written heavy_init might well need to use it. But in fact the existing heavy_init routines do make use of the signal.

For example, consider speedtch_upload_firmware in speedtch.c. It does two things: it sends a bunch of urbs to the modem, and it performs the above msleep_interruptible. If disconnect is called, any urbs in progress promptly fail and any newly submitted urbs fail at once; thus the only thing that can take an appreciable amount of time is the msleep_interruptible. But this will also exit at once because of the signal sent by usbatm during disconnect. So, in this case, the signal reduces the maximum time the USB subsystem is blocked in disconnect from one second to zero seconds.

Now consider firmware loading, a nasty case. This is also done in heavy_init, and can take an infinite amount of time if the firmware is not found (the user can choose an infinite timeout); the default timeout of 10 seconds is already plenty long. Firmware loading also needs to exit at once if the modem is disconnected. You may well wonder how speedtch_heavy_init arranges to cancel firmware loading when the signal comes in. The answer is that it doesn't cancel it. But this is not a reason to remove the sending of the signal, it is a reason to improve speedtch_heavy_init. This is not so easy, because the firmware subsystem doesn't give the kernel any way of cancelling a firmware load once started, which is why it doesn't happen right now.

Once you accept that a signal needs to be sent, you can't remove all those completions etc that your patch deleted, because it introduces races: they are there to make sure that (a) signals are unblocked in the thread before the signal can possibly be sent, and (b) the signal is not sent to the wrong thread if the kernel thread exits at a badly chosen moment and the pid is recycled. I believe the current setup is race free, but please don't hesitate to correct me. If you want to get rid of the pid and instead use a pointer to the thread then avoiding race (b) becomes even more important, since then rather than shooting down the wrong

thread you send a signal to a thread which no longer exists, surely a fatal mistake.

So it's a NACK for your current patch I'm afraid.

Best wishes,

Duncan Sands.

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers