ServoMaster: Rant on Serial/USB Bridges

Table of contents

1. The Subject

For the scope of discussion, let's define the `Serial/USB Bridge`, a.k.a. `Serial/USB Adapter`, as the device that has the following interfaces, ordered from controlled hardware to controlling application:

1. A physical serial interface, integrated or external - the controlled hardware is connected to this interface;
2. A physical USB interface which connects the bridge to the host computer;
3. A virtual serial interface that is, supposedly, undistinguishable from a physical serial interface - the controlling application is using this port.

2. The Scope

The rest of the article is dedicated to issues related to using devices equipped with Serial/USB bridges in the context of high uptime, low maintenance, mission critical applications.

The most important issue is the ability to gracefully recover from hardware failures without shutting down the entire system. Serial devices don't provide this ability at all, unless they're specifically engineered to do that - and in any case, even unplugging the serial device and plugging another one should be strongly discouraged, though it seems to be a common practice. On the other hand, you can just unplug a USB device and replace with an identical (or compatible) one.

3. The Shortcut

Apparently, Serial/USB bridges were thought to be a cool idea by embedded hardware designers - why, you get all the benefits of a USB interface with almost no extra effort - connect the chip and bingo, your piece of hardware now has a USB interface. Moreover, it is possible to retain a serial interface as well with a very little extra effort, so your hardware instantly becomes dual-interfaced.

On the other hand, I bet that driver programmers were equally ecstatic about this, at least at the beginning. Look, we don't have to write yet another driver - it works already!

4. The Problem

If it looks too good to be true, it usually is.

See, there's just this little pesky problem of the USB protocol being a protocol that supports *removable* devices. And you probably remember the days when you didn't dare to plug or unplug a serial device without shutting off the host computer first, lest you fry them both. The serial interface never even hinted at a possibility of the other side being removable - it only has a notion of a communication error, nothing else.

Coupled with the fact that the devices provided with the Serial/USB bridges quite often have limited intelligence, and some don't even provide any kind of status feedback (they're write-only)... Houston, we have a problem. Actually, several.

4.1. Electrical Incompatibility

USB devices, by necessity, have to be connected to the power circuit provided by the USB bus. Also, it may happen that a device originally designed as a serial device doesn't provide a proper ground separation between the USB provided ground and an external power provided for the rest of the device. Therefore, it is possible, at least theoretically, that a deficient hardware design may allow you to fry either itself, or the USB hub or the USB port it's plugged into.

4.2. Missing Capabilities Discovery Support

(Thanks to Jerry Scharf for reminding) USB devices are able to advertise their capabilities and functionality, to some extent (at least device class information is available). Serial ports do not have an ability to provide this information. "Probing" serial ports is ill-advised and unsafe - data sequences meaningful for one device may break another and cause massive damage on peripheral equipment.

4.3. Missing Support for Removable Devices

Your application will not be able to take advantage of the fact that USB specification supports the notion of a removable device.

4.4. Proprietary Drivers

Some of the Serial/USB bridges on the market chose not to publish their USB API. One pathetic case is Silicon Laboratories CP2101 - and unfortunately, it is used in one of USB controllers, which is pretty damn good otherwise.

The worst case consequence of this is that you won't be able to use the device with Linux, which kind of defeats the purpose (for me, at least).

4.5. Runaway Ports

Note:
This section is a total speculation and may be completely untrue, so let's just call it what it is, a speculation, until it is either confirmed or denied.

What happens to a virtual serial port when the actual hardware is disconnected? Furthermore, what happens to it when the hardware is reconnected? Is the serial port name the same?

What if I configure the application to use, say, a serial port `/dev/ttyS4` when I have the only Serial/USB bridge plugged in, and then reboot the box and add another Serial/USB bridge? Is the port name going to be the same? Likewise, if I unplug a device and plug in a different kind of a device, won't I get a situation when I'm talking not to the device I think I'm talking to?

5. The Solution

The only real (I mean *reliable*) solution, at least the way it looks from where I'm standing, is to ignore the virtual serial port drivers and write the driver directly against the USB specifications for the Serial/USB bridge. This way, we get the best of both worlds: it is possible to at least partially reuse the serial driver code, and it is possible to address the concerns addressed above, at least the removable device part and runaway port issue.

There are downsides to this solution, of course.

First of all, the resent trend in Linux kernel development seems to be to create kernel modules for all kinds of stuff imaginable, so I would assume, the modules will have to be removed - and you only pray that your favorite digital camera doesn't have the same kind of Serial/USB bridge chip as your favorite servo controller.

Then, it seems that capabilities discovery is still not possible to support, due to the fact that it is the bridge that makes itself visible to the USB bus, not the device controlled by the bridge.

And eventually, this solution implies more work for the software driver developer, and slower time-to-market, so you have to consider whether the problems outlined above matter for your cause.