

Re: pci error recovery procedure

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-09/msg00171.html>

- *From:* "Zhang, Yanmin" <yanmin_zhang@xxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 01 Sep 2006 17:04:09 +0800
-

On Fri, 2006-09-01 at 11:42, Zhang, Yanmin wrote:

On Fri, 2006-09-01 at 01:50, Linas Vepstas wrote:

On Thu, Aug 31, 2006 at 03:10:12PM +0800, Zhang, Yanmin wrote:

Linas,

I am reviewing the error handlers of e1000 driver and got some ideas. My startpoint is to simplify the err handler implementations for drivers, or driver developers are **not willing** to add it if it's too complicated.

I don't see that its to complicated ...

Originally, I didn't think so, but after I try to add err_handlers to some drivers, I feel it's too complicated.

1) Callback mmio_enabled looks useless.
Documentation/pci-error-recovery.txt
says the current powerpc implementation does not implement this callback.

I don't know if its useless or not. I have not needed it yet for the symbios, ipr and e1000 drivers, but its possible that some more sophisticated device may want it. I'm tempted to keep it a while longer before discarding it.

The scenario is this: the device driver decides that, rather than asking for a full electrical reset of the card, instead, it wants to perform its own recovery. It can do this as follows:

Re: pci error recovery procedure

- a) enable MMIO
- b) issue reset command to adapter
- c) enable DMA.

If we enabled both DMA and MMIO at the same time, there are many cases where the card will immediately trap again -- for example, if its DMA'ing to some crazy address. Thus, typically, one wants DMA disabled until after the card reset. Without the `mmio_enabled()` reset, there is no way of doing this.

The new `error_resume`, or the old `slot_reset` could take care of it. The specific device driver knows all the details about how to initiate the devices. The `error_resume` could call the step a) b) c) sequentially while doing checking among steps.

If there is really a device having specific requirement to reinitiate it (very rarely), it could use walkaround, such like schedule a `WORKER`. No need to provide a generic `mmio_enabled`.

2) Callback `slot_reset` could be merged with `resume`. The new `resume` could be:
`int (*error_resume)(struct pci_dev *dev);` I checked `e1000` and `e100` drivers and think there is no actual reason to have both `slot_reset` and `resume`.

The idea here was to handle multi-function cards. On a multi-function card, **all** devices need to indicate that they were able to reset. Once all devices have been successfully reset, then operation can be resumed. If the reset of one function fails, then operation is not resumed for any of the functions.

I don't think we need `slot_reset` to coordinate multi-function devices. The new `error_resume` could take care of multi-function card. 'reset' here means driver need do I/O to detect if the device (function) still works well. If a function of a multi-function device couldn't reset while other functions could reset, other functions could just go on to reinitiate. In the end, the error recovery procedure (`handle_eeh_events` in PowerPC implementation) could check all the returning values of `error_resume`. If there is a failure value, then removes all the functions' `pci_dev` of the device from the bus.

3) `link_reset` is not used in pci express aer implementation, so it could be deleted also.

Re: pci error recovery procedure

OK. Link reset was added explicitly to support PCI-E, so if its not wanted, we can eliminate it.

How did you test e1000 err_handler?

We have three methods (I thought these were documented). In one, a technician brushes a grounding strap to some of the signal pins. In the second, slots are populated with known-bad cards. The third test involves sending a command down to the pci bridge chip, telling it to behave as if it detected an error. For development, the last is quick-n-easy.

Thanks for your explanation.

In the simulated enviroment, the testing might be incorrect.

Why would it be incorrect? I mean, we don't simulate having someone pour a cup of coffee into the guts of the machine ... but my understanding is the machines do get standard vibration/thermal/humidity testing, which is good enough for me.

For example, e1000_io_error_detected would call e1000_down to reset NIC.

Why would that be incorrect?

During our last discussion on LKML, you said PowerPC will block further I/O if the platform captures a pci error, so the all I/O in e1000_down will be blocked. Later on, e1000_io_slot_reset will reenable pci device and initiate NIC. I guess late initiate might fail because prior e1000_down I/O don't reach NIC.

Why would it fail? The e1000_down serves primarily to get the Linux kernel into a known state. It doesn't matter what happens to the card, since the next step will be to perform an electrical reset of the card.

Re: pci error recovery procedure

Who will perform the electrical reset of the card? Function `e1000_reset` or the platform? If it's the platform, I agree with you, but if it's `e1000_reset`, it might not work because `e1000_reset` uses a `e1000`-specific approach to reset the card. I'm not sure if the `e1000_reset` will restore the NIC to fresh system power-on state. At least, from the source codes, `e1000_reset` couldn't.

One more comment: The second parameter of `error_detected` also could be deleted because recovery procedures will save error to `pci_dev->error_state`.

So, the `err_handler` `pci_error_handlers` could be:

```
struct pci_error_handlers
{
pci_ers_result_t (*error_detected)(struct pci_dev *dev);
pci_ers_result_t (*error_resume)(struct pci_dev *dev);
};
```

Yanmin

—

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

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

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