Deferred IO

Deferred IO is a way to delay and repurpose IO. It uses host memory as a buffer and the MMU pagefault as a pretrigger for when to perform the device IO. The following example may be a useful explanation of how one such setup works:

- userspace app like Xfbdev mmaps framebuffer
- deferred IO and driver sets up fault and page_mkwrite handlers
- userspace app tries to write to mmaped vaddress
- we get pagefault and reach fault handler
- fault handler finds and returns physical page
- we get page_mkwrite where we add this page to a list
- schedule a workqueue task to be run after a delay
- app continues writing to that page with no additional cost. this is the key benefit.
- the workqueue task comes in and mkcleans the pages on the list, then completes the work associated with updating the framebuffer. this is the real work talking to the device.
- app tries to write to the address (that has now been mkcleaned)
- get pagefault and the above sequence occurs again

As can be seen from above, one benefit is roughly to allow bursty framebuffer writes to occur at minimum cost. Then after some time when hopefully things have gone quiet, we go and really update the framebuffer which would be a relatively more expensive operation.

For some types of nonvolatile high latency displays, the desired image is the final image rather than the intermediate stages which is why it's okay to not update for each write that is occurring.

It may be the case that this is useful in other scenarios as well. Paul Mundt has mentioned a case where it is beneficial to use the page count to decide whether to coalesce and issue SG DMA or to do memory bursts.

Another one may be if one has a device framebuffer that is in an usual format, say diagonally shifting RGB, this may then be a mechanism for you to allow apps to pretend to have a normal framebuffer but reswizzle for the device framebuffer at vsync time based on the touched pagelist.

How to use it: (for applications)

No changes needed. mmap the framebuffer like normal and just use it.

How to use it: (for fbdev drivers)

The following example may be helpful.

1. Setup your structure. Eg:

```
static struct fb_deferred_io hecubafb_defio = {
        .delay          = HZ,
        .deferred_io    = hecubafb_dpy_deferred_io,
};
```

The delay is the minimum delay between when the page_mkwrite trigger occurs and when the deferred_io callback is called. The deferred_io callback is explained below.

2. Setup your deferred IO callback. Eg:

```
static void hecubafb_dpy_deferred_io(struct fb_info *info,
                                struct list_head *pagelist)
```

The deferred_io callback is where you would perform all your IO to the display device. You receive the pagelist which is the list of pages that were written to during the delay. You must not modify this list. This callback is called from a workqueue.

3. Call init

```
        info->fbdefio = &hecubafb_defio;
        fb_deferred_io_init(info);
```

4. Call cleanup

```
        fb_deferred_io_cleanup(info);
```