The io_mapping functions

API

The io_mapping functions in linux/io-mapping.h provide an abstraction for efficiently mapping small regions of an I/O device to the CPU. The initial usage is to support the large graphics aperture on 32-bit processors where ioremap_wc cannot be used to statically map the entire aperture to the CPU as it would consume too much of the kernel address space.

A mapping object is created during driver initialization using:

```
struct io_mapping *io_mapping_create_wc(unsigned long base,
                                        unsigned long size)
```

'base' is the bus address of the region to be made mappable, while 'size' indicates how large a mapping region to enable. Both are in bytes.

This _wc variant provides a mapping which may only be used with io_mapping_map_atomic_wc(), io_mapping_map_local_wc() or io_mapping_map_wc().

With this mapping object, individual pages can be mapped either temporarily or long term, depending on the requirements. Of course, temporary maps are more efficient. They come in two flavours:

```
void *io_mapping_map_local_wc(struct io_mapping *mapping,
                              unsigned long offset)

void *io_mapping_map_atomic_wc(struct io_mapping *mapping,
                               unsigned long offset)
```

'offset' is the offset within the defined mapping region. Accessing addresses beyond the region specified in the creation function yields undefined results. Using an offset which is not page aligned yields an undefined result. The return value points to a single page in CPU address space.

This _wc variant returns a write-combining map to the page and may only be used with mappings created by io_mapping_create_wc()

Temporary mappings are only valid in the context of the caller. The mapping is not guaranteed to be globaly visible.

io_mapping_map_local_wc() has a side effect on X86 32bit as it disables migration to make the mapping code work. No caller can rely on this side effect.

io_mapping_map_atomic_wc() has the side effect of disabling preemption and pagefaults. Don't use in new code. Use io_mapping_map_local_wc() instead.

Nested mappings need to be undone in reverse order because the mapping code uses a stack for keeping track of them:

```
addr1 = io_mapping_map_local_wc(map1, offset1);
addr2 = io_mapping_map_local_wc(map2, offset2);
...
io_mapping_unmap_local(addr2);
io_mapping_unmap_local(addr1);
```

The mappings are released with:

```
void io_mapping_unmap_local(void *vaddr)
void io_mapping_unmap_atomic(void *vaddr)
```

'vaddr' must be the value returned by the last io_mapping_map_local_wc() or io_mapping_map_atomic_wc() call. This unmaps the specified mapping and undoes the side effects of the mapping functions.

If you need to sleep while holding a mapping, you can use the regular variant, although this may be significantly slower:

```
void *io_mapping_map_wc(struct io_mapping *mapping,
                        unsigned long offset)
```

This works like io_mapping_map_atomic/local_wc() except it has no side effects and the pointer is globaly visible.

The mappings are released with:

```
void io_mapping_unmap(void *vaddr)
```

Use for pages mapped with io_mapping_map_wc().

At driver close time, the io_mapping object must be freed:

```
void io_mapping_free(struct io_mapping *mapping)
```