dm-io

Dm-io provides synchronous and asynchronous I/O services. There are three types of I/O services available, and each type has a sync and an async version.

The user must set up an io_region structure to describe the desired location of the I/O. Each io_region indicates a block-device along with the starting sector and size of the region:

```
struct io_region {
   struct block_device *bdev;
   sector_t sector;
   sector_t count;
};
```

Dm-io can read from one io_region or write to one or more io_regions. Writes to multiple regions are specified by an array of io_region structures.

The first I/O service type takes a list of memory pages as the data buffer for the I/O, along with an offset into the first page:

```
struct page_list {
   struct page_list *next;
   struct page *page;
};

int dm_io_sync(unsigned int num_regions, struct io_region *where, int rw,
               struct page_list *pl, unsigned int offset,
               unsigned long *error_bits);
int dm_io_async(unsigned int num_regions, struct io_region *where, int rw,
                struct page_list *pl, unsigned int offset,
                io_notify_fn fn, void *context);
```

The second I/O service type takes an array of bio vectors as the data buffer for the I/O. This service can be handy if the caller has a pre-assembled bio, but wants to direct different portions of the bio to different devices:

```
int dm_io_sync_bvec(unsigned int num_regions, struct io_region *where,
                    int rw, struct bio_vec *bvec,
                    unsigned long *error_bits);
int dm_io_async_bvec(unsigned int num_regions, struct io_region *where,
                     int rw, struct bio_vec *bvec,
                     io_notify_fn fn, void *context);
```

The third I/O service type takes a pointer to a vmalloc'd memory buffer as the data buffer for the I/O. This service can be handy if the caller needs to do I/O to a large region but doesn't want to allocate a large number of individual memory pages:

```
int dm_io_sync_vm(unsigned int num_regions, struct io_region *where, int rw,
                  void *data, unsigned long *error_bits);
int dm_io_async_vm(unsigned int num_regions, struct io_region *where, int rw,
                   void *data, io_notify_fn fn, void *context);
```

Callers of the asynchronous I/O services must include the name of a completion callback routine and a pointer to some context data for the I/O:

```
typedef void (*io_notify_fn)(unsigned long error, void *context);
```

The "error" parameter in this callback, as well as the **error* parameter in all of the synchronous versions, is a bitset (instead of a simple error value). In the case of an write-I/O to multiple regions, this bitset allows dm-io to indicate success or failure on each individual region.

Before using any of the dm-io services, the user should call dm_io_get() and specify the number of pages they expect to perform I/O on concurrently. Dm-io will attempt to resize its mempool to make sure enough pages are always available in order to avoid unnecessary waiting while performing I/O.

When the user is finished using the dm-io services, they should call dm_io_put() and specify the same number of pages that were given on the dm_io_get() call.