# Device Driver Manual

By: Eric Byjoo

## Summary

The purpose of this library you are currently using is to convert a linear device or a set of files into a usable product or output. An example would be a set of disks you would like to manipulate. For the sake of this manual, we will refer to it as a device. This software is highly capable and contains multiple features. One big feature is that it supports caching, making reading your device and writing to it more efficient. We also support using servers so that you can utilize this library throughout the internet and work remotely. As you proceed through this manual, be weary of the parameters so that you don’t run into errors.

## Before Starting

Before importing this library, have the desired device to read inside the correct directories. If you want to write to the device, have that in the same directories. Finally, be sure to have something to compile your device. For the purpose of this manual, it is listed as a makefile, and then make is referred to compile your device.

Additionally, please install the libssl-dev library to ensure compatibility with this library. The Linux command for this would be:

sudo apt install libssl-dev

All of the function calls going forward will look like the following:

mdadm_example(parameters)

Mdadm stands for multiple disk and device driver administration. This is the implementation that this library has implemented into Linux.

## Mounting

For this library to prepare all the commands from your device, we need to mount it. This will be done by calling.

mdadm_mount();

If a value of 1 is returned, the device is successfully mounted, and you may proceed. If this function returns -1, an error has occurred. This error is because you are attempting to mount the device when it has already been mounted.

## Reading And Writing The Files

To read the device you want, you may call the following command.

mdadm_read(uint32_t address, uint32_t len, uint_t 8 buf);

To write the device you want, you may call the following command.

mdadm_write(uint32_t address, uint32_t len, uint_t 8 buf);

The address parameter refers to the desired address to read from or write to. The len parameter is the desired number of bytes you want to read from or write. The buf parameter specifies the block of information you desire to read from or write to.

If calling these functions returns the length of bytes you desired to read or write, it has been completed successfully. However, if it returns -1, there has been an error attempting to read the device.

Instances this function will have an error:

1. Not having the device mounted
2. Attempting to read or write more than 1024 bytes or approximately 1KB
3. Attempting to read from an address not possible on your device
    a. This includes starting at a valid address and attempting to read onwards
    b. Examples of failure for a file with a max address number of 1,048,576
        i. mdadm_read(1048500,80,buf)
        ii. mdadm_read(1050000,100,buf)
        iii. mdadm_read(-1,100,buf)
        iv. mdadm_write(1048500,80,buf)
        v. mdadm_write(1050000,100,buf)
        vi. mdadm_write(-1,100,buf)
4. Attempting to read from an empty buffer (NULL) or write of a length to an empty buffer (NULL) of your device

## Unmounting

Once you have completed your process of reading and/or writing your device, it is important to unmount the device, even if you are not looking to start a new one, to prevent memory leaks. To do this, simply call the command below.

mdadm_unmount();
Calling this command should return a value of 1 for success. If it returns -1, there has been an error, meaning that you are attempting to unmount a device that has already been mounted.

# Caching Support

This library supports caching for both reading and writing your files. Caching is an important aspect that has helped this library reduce latencies. By enabling the cache, you will notice a dramatic increase in reading and writing on large devices, especially if you utilize a large cache.

This caching system works by searching the block that corresponds to the address you have called in read or write and checking if it is in the cache. If it's there, it can just take it from the cache without reducing reading or writing speed. It will read and insert into the cache using a least recently used policy if it isn't.

## Starting the cache

To enable the cache, simply call the following command.

mdadm_cache_create(int number_of_entries)

The number of entries refers to the size of the cache you want to create. This function will return one on success. However, if it returns -1, an error has been defined here.

Instances creating the cache will fail:

1. Attempting to create a cache when a cache already exists
2. Attempting to create a cache of less than two entries
3. Attempting to create a cache with more than 4096 entries
4. Attempting to read or write a portion of your device with a buf that is empty (NULL) while using the cache will cause an error
5. Attempting to read or write from a place non-existent in your device while using the cache will cause an error

## Ending the cache

It is important to destroy your cache after you have finished using it. Without destroying it, you cannot create a new cache, and you may cause memory leaks. The following command will destroy the cache.

mdadm_cache_destory()

This call will return a value of 1 to indicate success. If it returns -1, you have attempted to destroy a cache when there is none.

# Networking Support

Networking is a feature of this library that allows you to read or write to various places such as data centers.

## Connecting

To connect to your desired server, simply run the following command.

mdadm_connect(const char *ip, uint16_t port)

The *ip parameter refers to the IP address you want to access, and the port parameter refers to the port your server listens to information from.

Important notes

1. This connectivity support is for the IPV4 protocol, is your domain is using an IPV6 protocol connecting will not work
2. The communication semantics is using TCP, which is a stream; it does not use a datagram
3. If desired, you may use the cache along with the network support, but they are not dependent on each other

Instances of errors:

1. If calling connect triggers a statement saying “Address or Port error,” then it is likely there has been a problem with the IP address or port address given. This instance will also make the call return -1.
2. If calling connect triggers a statement saying “connect error,” then there have been some other issues attempting to connect to your desired server. This instance will also make the call return -1.
3. If attempting to make a call is non-existent in the library, the network will state there is an error and return -1

## Disconnecting

It is important to disconnect from the server once completed to prevent memory bugs and not accidentally send more information than needed. To disconnect, simply run the following command.

jbod_disconnect()

If you call it, this function will return -1 while no server is connected.