mdadm Manual

By Kevin Starnes

Introduction:

The manual has eight functions that can do many operations. It can reduce latency, help set up storage space, and increase the flexibility of the system using networking. It is able to read from many disks, and is able to write to this disks as well. Next it supports caching by saving the most often used data that is saved in a secondary storage by using the LRU algorithm. This will increase the performance. Finally, it is able to connect to a JBOD server remotely, and call JBOD commands over a network.

The list below are the eight functions within this API. Please note, some functions require parameters and some do not. There are some hidden helper functions that help these eight functions work smoothly. All of the functions work on linear devices. Overall, these functions are the only ones that will need to be called.

Library Functions:

- mdadm_mount()
- mdadm_unmount()
- mdadm_read()
- mdadm_write()
- mdadm_cache_create()
- mdadm_cache_destroy()
- mdadm_connect()
- mdadm_disconnect()

Usage:

mdadm_mount(void):

This function should always be called first on the JBOD. It will mount the linear device and will allow for all other functions to be called. It will allow the user to call all functions below this one. If it is not mounted first, then all other commands will not be able to work. If the linear device is already mounted, then calling mdadm_mount() again will not work and will return the value -1. This function will return the value 1 on success if it was mounted correctly .

mdadm_unmount(void):

This works just like mdadm_mount, but is flipped. This function should always be called last on the JBOD. It will unmount the linear device and will not allow for any functions to be called after this function. It will return the value 1 if it successfully unmounted the linear device. It will return the value -1 if it failed, or if mdadm_unmount() was already called.

mdadm_read(uint32_t addr, uint32_t len, uint8_t *buf):

This function's purpose is to read the contents from blocks within the disks. This function is able to read within the block, across blocks, across multiple blocks, and across multiple disks. There are three parameters for mdadm_read(). The first one is the addr, or known as the address. This will be the start address from where the user wants to read from. Next is the len, or known as the length. The maximum value for this is 1024 bytes. Len should never be 0 as well. The third parameter is the buf, or known as the buffer. The maximum value of buf is 1,048,576 bytes. This value takes the sum of addr and len. If values of the sum is over 1,048,576, then this function will fail. Buf should never be empty.

Example code:

```
mdadm_read(0x760, 300, 300)
```

mdadm_write(uint32_t addr, uint32_t len, uint8_t *buf):

This function's purpose is to write content to blocks within the disks. This function is able to write within the block, across blocks, across multiple blocks, and across multiple disks. There are three parameters for mdadm_write(). The first one is the addr, or known as the address. This will be the start address from where the user wants to write from. The maximum value for the address should be at most . Next is the len, or known as the length. The maximum value for this is 1024 bytes. Len should never be 0 as well. The third parameter is the buf, or known as the buffer. The buf is the data that is being written into the block. The maximum value of buf is 1,048,576 bytes. This value takes the sum of addr and len. If values of the sum is over 1,048,576, then this function will fail.

Example code:

mdadm_read(0x760, 256, 0x10)

mdadm_cache_create(int num_entries):

This function is needed if the user would like the improved performance due to read() caching. This function is used to create a certain amount of cache entries the user wishes to use. It uses the parameter num_entries which takes an integer. The number of entries must be at least two or at most 4096. Anything over or under will cause it to fail. After calling this function, you must call mdadm_cache_destroy to avoid memory leaks. Calling this function twice without calling mdadm_cache_destroy will return the value -1, meaning it failed to create the cache. If it was successful on creating the cache, then it will return the value 1.

Example code:

mdadm_cache_create(2048)	mdadm_cache_create(10000)
This will create 2,048 entries.	This will fail to create since its over 4096 entries

mdadm_cache_destroy(void):

This function is used to free the space allocated by the mdadm_cache_create() function. Calling this function twice without calling mdadm_cache_create will return the value -1, meaning it failed. If it was successful on destroying the cache, then it will return the value 1. This function must be called to avoid memory leaks.

mdadm_connect(const char *ip, uint16_t port):

Instead of using JBOD commands to a locally attached JBOD system, this function will allow the user to use a server base JBOD system. This function takes two parameters. The first one is the *ip. It is a char type variable, so the user will have put quotes ,“ip“, around the IP address the user wishes to connect to. The next parameter is the port the user wishes to use to send and receive data. It will return “true” if it successfully connects to the JBOD server, and “false” if it does not successfully connect to the JBOD server. Please note, the IP address must be a IPv4 address, and the user must call mdadm_disconnect() after calling this function to avoid memory leaks.

Example code:

```
mdadm_connect("127.0.0.1", 3333)
```

mdadm_disconnect(void):

This function is used to disconnect from the JBOD server. It does not return anything. It’s a simple command that will only work if the user is connected to a JBOD server already. This function must be called, after mdadm_connect(), to avoid memory leaks.