MDADM Linear Storage System With Networking Applications

System Overview

This storage system is based on 256 disks, which each contain 256 blocks with each block containing 256 bytes of data. The total space for this system (JBOD for short) is 1,048,576 bytes or 1 MB, which corresponds to 65,536 bytes in each disk. This library also has implemented caching to improve performance and for the reduction of latency. The system works with remote JBOD servers in order to allow for client / server connections. This allows for users to use it remotely to store and read their desired data.

Public Interface: Preparing the System

In order for the system to begin, call the mdadm_mount() function of which has no in parameters. This function mounts all the disks and prepares the system to receive calls. Before calling this function, all other commands will fail. On the other end, mdadm_unmount() is the counterpart to mdadm_mount(). Unmount takes all of the disks off of the JBOD and prepares the system to turn off. Important note: calling mdadm_mount() consecutively will fail without a mdadm_unmount() between the mount commands. Similarly, calling mdadm_unmount() consecutively without a mdadm_mount() in between them will also cause failure.

Public Interface: Setting up network application

To allow for client and network communications, the user must call two functions. To begin with, the user will call jbod_connect(const char *ip, uint16_t port). This function takes in an ip address and a port number that the client wants to connect to. This connection allows for the client to send request messages to the JBOD servers and also for the receiving of messages from JBOD. This function needs to be called before any request messages are sent to the server or else it will result in failure. Once the user is finished with their requests, issue the

jbod_disconnect() command which will close the connection. Failure to call this command can cause issues with the system and faulty usage.

Public Interface: Data Reading and Writing

The system allows the user to read pre-existing data and write new data into the system. To reiterate, these functions will not be able to be performed without mdadm_mount() being called beforehand.

MDADM_READ

The mdadm_read(uint32_t address, uint32_t length, uint8_t * buffer) function is the call for reading data from the system. To use this function, the first input corresponds to the given address where the user wants to begin reading from. The second input is an integer value that corresponds to the amount of bytes that you want to be read. The value of address + length CANNOT be greater than 1,048,576 or else the system will fail. Also, the length cannot exceed 1,024 as the system limits the size of buffers to this value. Furthermore, the buffer cannot be NULL if the user is requesting to read any bytes (length>0) Lastly, the final input is a buffer specified by the user, this cannot be larger than 256 bytes.. Overall, the mdadm_read function reads *Length* bytes starting from *address* into the *buffer.*

MDADM_WRITE

The mdadm_write(uint32_t address, uint32_t length, const uint8_t * buffer) function is the counterpart to the read function. The write function will write length bytes from the buffer, into the current DISKID /BLOCKID specified by the value address. Similarly to the read function, length cannot exceed 1,024, length+address cannot be greater than 1,048,576, and the buffer cannot be NULL if the user is trying to write data (length>0).

Public Interface: Cache

The library also has caching options to improve latency and reduce the costs of running the system. Using the cache is optional, however creating a large cache can greatly reduce the cost and will enhance the performance of the system.

CACHE_CREATE

To create the cache, call the command cache_create(int num_entries). The num_entries parameter corresponds to how large you want the cache to be. On large reads and writes, larger caches are more desired and with smaller reads / writes, a smaller cache is sufficient. This command cannot be called twice without a cache_destroy() command in between. The cache has a minimum size of 2 entries and a maximum size of 4,096 entries. Anything outside of this range will result in failure.

CACHE_DESTROY

In order to prevent memory leaks from the system, calling the cache_destroy() command when done with the system is essential. Failure to call this command will result in a memory leak and system failure. Cache_destroy() takes in no commands and will delete the cache. Calling this command with no current cache created will result in failure.