TECHNICAL REPORT

NAME: Vishnu Sree Peram

ACCESS ID: VJP5266@PSU.EDU

MDADM MODULE

Mdadm stands for multiple disk and device administration, and it is a tool for making optimal use of multiple disks in a computer. Some of these uses include reading data from or writing data to a specific disk and block. The functions below will guide the user to appropriately use the capabilities of multiple disks to their full extent. Description of each function will be accompanied by a snippet of code in C language for further clarification.

This implementation of MDADM also provides cache and network features to accommodate the user's needs in various instances. A detailed explanation of these features and the functions they provide is provided by the end of this document.

mdadm_mount()

This function does not take any arguments and returns 1 on success and -1 otherwise. Upon calling this function, all the jbod disks will be mounted and ready to be operated on. It is worth noting that any other functions in this module that are called prior to making a call to this function will be void.

Sample code:

```
if (mdadm_mount() == 1) {
   printf("disks are successfully mounted");
}
```

mdadm_unmount()

This function does not take any parameters and returns 1 on success and -1 on failure. A call to this function will unmount all the disks and after that, the user will be unable to perform any operations on the disks. If the user wishes to perform more operations, they will have to mount the disks again.

Sample code:

```
if (mdadm_unmount() == 1) {
   printf("disks are successfully unmounted");
}
```

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

addr parameter takes address as a 32-bit unsigned integer, len parameter takes length of the bytes to read as a 32-bit unsigned integer, and buf takes a pointer to a buffer with each slot being able to accommodate an 8-bit unsigned integer and of a length of minimum len bytes. This functon copies len number of bytes starting from addr into buf. It returns len on success and -1 on failure.

Invalid parameters are as follows:

- len is greater than 1024.
- addr is greater than 1048576
- buf is null
- sum of addr and len is greater than 1048576

Sample code:

```
uint32_t addr = 420;
uint32_t len = 258;
uint8_t *mybuf[258];
if (read(addr, len, mybuf) == len) {
   printf("Success!");
}
```

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

addr parameter takes address as a 32-bit unsigned integer, len parameter takes length of the bytes to write as a 32-bit unsigned integer, and buf takes a pointer to a buffer containing the contents to be written to the disks. This function will write the first len number of bytes from buf to the disks starting at the address addr. It returns len on success and -1 on failure.

Invalid parameters are as follows:

- len is greater than 1024.
- addr is greater than 1048576.
- buf is null.
- sum of addr and len is greater than 1048576.

Sample code:

```
uint32_t addr = 420;
uint32_t len = 258;
uint8_t *mybuf[258];

// populate mybuf

if (write(addr, len, mybuf) == len) {
    printf(“Success!”);
}
```

CACHE

Cache is a part of the memory hierarchy and allows the user to read from the memory faster with uncompromised accuracy. This can be especially beneficial if data has to be read from the disks multiple times. The following descriptions of the functions will give the user information about how to use the cache.

cache_create(int num_entires)

The num_entries parameter take an integer and the function itself returns 1 on success and -1 on failure. This function creates a cache with enough space to store num_entries number of entries in it. Note that without calling this function, cache cannot be used and calling it again without destroying the initial cache will result in a failure.

Sample code:

```
int entries = 256;
If (cache_create(256) == 1) {
    printf(“successfully created the cache!”);
}
```

cache_destroy()

This function does not take any parameters, returns 1 on success and -1 on failure. The only instance when this function fails is when there is no cache to be destroyed i.e, when a cache haven’t been created.

Sample code:

```
If (cache_destroy() == 1) {
    printf(“cache has been destroyed”);
}
```

NETWORK CAPABILITIES

This feature will enable the user to read the data from and write the data to the servers that are located elsewhere.

jbod_connect(const char *ip, uint16_t port)

This function will take a character pointer to the ip address and a 16-bit unsigned integer for port number and returns boolean true on success, false on failure. The purpose of this function is it establishes connection to the server, on which the mdadm operations are performed.

Sample code:

```
// define ip
// define port number
if (jbod_connect(*ip, port) == 1) {
    printf(“successfully connected to the server!”)
```

}

jbod_disconnect()

This function does not take any arguments and returns nothing after completion. The purpose of this function is to close the established connection to the server.

Sample code:

```
// establish a connection
// perform necessary operations
jbod_disconnect();
```