A Basic Guide to CMPSC 311's JBOD

Pennsylvania State University
State College, PA

Viswa Rathnakumar

Table of Contents

Introduction

Welcome to the basic JBOD manual. JBOD, or “just a bunch of disks”, is a way to describe multiple memory disks acting as one big module. JBOD is best used for anyone looking to set up quick and efficient memory storage with less importance to security. In other words, JBOD helps maintain how to write and read to memory across different modules to keep user code efficient and clean. In this implementation of JBOD, we support linking 16 disks made up of 256 blocks each. The blocks are each 256 bytes, so each disk is 65,536 bytes. In total, the system contains 16 x 256 x 256 = 1048576 bytes, or 1 MB. In addition, our JBOD library supports from 2 to 4096 bytes caching for better performance when reading and writing to the JBOD. The library also supports working with remote JBOD servers, for any of your work-from-home long-distance needs.

Getting Started

Mounting

To start using the JBOD, we need to mount all the linear memory disks with this command.

```
int mdadm_mount(void)
```

The command returns 1 if successful, and -1 if the JBOD is already mounted. All other commands will not work until the JBOD is mounted.

Now that the JBOD is mounted, let's start writing and reading the JBOD!

Reading

We can read at most 1024 bytes at a time from JBOD with a linear address from 0 to 65,634. The bytes read will be stored in the inputted buffer. Success returns 0 and failure returns -1.

Use this command to read to JBOD

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

Naturally, the inputted buffer of empty data must be larger than len. Be careful to not read out of bounds! This can be checked:

```
If (len + addr >= 1048576){
    return false;
}
```

It is important to note that passing a NULL buffer and a length greater than 0 will fail, but passing a NULL buffer and a length equal to 0 will not fail.

Writing

We can write at most 1024 bytes at a time into JBOD with a linear address from 0 to 65,634. The bytes written will be taken from the inputted buffer. JBOD can write across blocks and disks without any issues. Success returns 0 and failure returns -1.

Use this command to write to JBOD

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

Similarly to `mdadm_read`, the input buffer must be larger than len. To avoid writing out of bounds, check the parameters:

```
If (len + addr >= 1048576){
    return false;
}
```

It is important to note that passing a NULL buffer and a length greater than 0 will fail, but passing a NULL buffer and a length equal to 0 will not fail.

Unmounting

When we finish with our workload, we unmount the JBOD with this command.

`int mdadm_unmount(void)`

The command returns 1 if successful, and -1 if the JBOD is already unmounted. All other commands will not work after the JBOD is unmounted.

Example

We first mount the JBOD:
`mdadm_mount(void);`

Then, we can give any amount of read and write commands:

Input an empty buffer when reading. Make sure the buffer is larger or equal to the length. Avoid reading out of bounds with the given code in the Reading section.
`mdadm_read(uint32_t 0, uint32_t 511, uint8_t *buf);`

Input a buffer with your data when writing. Make sure the buffer is larger or equal to the length. Avoid writing out of bounds with the given code in the Writing section.
`mdadm_write(uint32_t 54343, uint32_t 1024, const uint8_t *buf);`

Finally, when we are done with the JBOD, we can unmount the disks with this command.
`mdadm_unmount(void);`

Caching with JBOD

The cache implemented with JBOD will improve performance for both reading and writing. The look aside cache is a fully associative cache with a LRU eviction policy. The blocks stored in the cache are key: JBOD Disk and JBOD block, value: block data. The bigger the cache created, from 2 up to 4096 bytes, the larger the performance gain. Note, caches may benefit in performance if they are “warmed up”. This means adding a few trivial read or write commands with data and addresses that will be reused in the working set. That way the cache will have blocks inserted and we can eliminate cold misses. We recommend using a cache for your work! All functions return 0 on success and -1 on fail.

Creating a Cache

Use this command to create a cache of size num_entries, from 2 to 4096 bytes. Success returns 0 and failure returns -1.

```
int cache_create(int num_entries)
```

Creating a cache that has been created will fail. You can create a cache in the middle of a workload between read and write commands, the JBOD will function with the newly created cache. The cache will manage eviction and insertion on its own, so users like you can reap the benefits!

Destroying the Cache

Use this command to destroy the created cache. Success returns 0 and failure returns -1.

```
int cache_destroy(void)
```

Destroying a cache that already has been destroyed will fail. You can destroy a cache in the middle of a workload between read and write commands, the JBOD will just function without a cache.

Example

If you haven’t already, mount the JBOD. Then create a cache with a size between 2 and 4096 bytes

```
mdadm_mount(void);
cache_create(4096);
```

We can now run the normal read and write commands to JBOD.

```
mdadm_read(uint32_t 0, uint32_t 511, uint8_t *buf);
mdadm_write(uint32_t 54343, uint32_t 1024, const uint8_t *buf);
```

If we wish to stop using a cache, call this command.

```
cache_destroy(void);
```

If you are done with JBOD, be sure to unmount.

```
mdadm_unmount(void);
```

Remote JBOD Server

The connection between the JBOD server and your machine is created with the server's IPv4 address and the server's port number. The connection is TCP. After connecting, the normal read and write commands can be given. The code internally handles sending instructions to the remote server.

Connecting to Remote JBOD Server

Use this command to create a connection to the JBOD server with the server's IPv4 address and port number. Success returns 0 and failure returns -1.

```
bool jbod_connect(const char *ip, uint16_t port)
```

Creating a connection when a connection exists will fail. JBOD commands executed after this will function with the remote JBOD server, given that the JBOD is first sent a mount command.

Disconnecting to Remote JBOD Server

Use this command to disconnect from the remote JBOD server. If the JBOD is already disconnected, the function will return regardless.

```
void jbod_disconnect(void)
```

If you are the last to use the JBOD before disconnecting, be sure to unmount the JBOD before disconnecting.

Example

Connect to the remote JBOD server with the correct ipv4 address and port number of the remote server.

```
jbod_connect(“228.6.23.238”, 6280);
```

Be sure to mount the JBOD.

```
mdadm_mount(void);
```

We can now run the normal read and write commands to JBOD.

```
mdadm_read(uint32_t 0, uint32_t 511, uint8_t *buf);
```

```
mdadm_write(uint32_t 54343, uint32_t 1024, const uint8_t);
*buf);
```

If you are done with JBOD, be sure to unmount.

```
mdadm_unmount(void);
```

Finally, disconnect from the remote JBOD server

```
jbod_disconnect(void);
```