Assignment #5 – `mdadm` Linear Device (Networking)
CMPSC311 - Introduction to Systems Programming
Fall 2021 - Prof. Aghayev

**Due date: April 30, 2021 (11:59 PM) EST**
**NO EXTENSIONS OR LATE SUBMISSIONS ACCEPTED**

Adding a cache to your `mdadm` system has significantly improved its latency and reduced the load on the JBOD. Before you finish your internship, however, the company wants you to add networking support to your `mdadm` implementation to increase the flexibility of their system. The JBOD systems purchased by the company can accept JBOD operations over a network using a proprietary networking protocol. Specifically, a JBOD system has an embedded server component that can be configured to have an IP address and listen for JBOD operations on a specific port. In this final step of your assignment, you are going to implement a client component of this protocol that will connect to the JBOD server and execute JBOD operations over the network. As the company scales, they plan to add multiple JBOD systems to their data center. Having networking support in `mdadm` will allow the company to avoid downtime in case a JBOD system malfunctions, by switching to another JBOD system on the fly.

Currently, your `mdadm` code has multiple calls to `jbod_operation`, which issue JBOD commands to a locally attached JBOD system. In your new implementation, you will replace all calls to `jbod_operation` with `jbod_client_operation`, which will send JBOD commands over a network to a JBOD server that can be anywhere on the Internet (but will most probably be in the data center of the company). You will also implement several support functions that will take care of connecting/disconnecting to/from the JBOD server.

## Protocol

The protocol defined by the JBOD vendor has two messages. The JBOD *request message* is sent from your client program to the JBOD server and contains an opcode and a buffer when needed. The JBOD *response message* is sent from the JBOD server to your client program and contains an opcode and a buffer when needed. Both messages use the same format:

| Bytes | Field | Description |
|---|---|---|
| 0-1 | length | The size of the packet in bytes |
| 2-5 | opcode | The opcode for the JBOD operation |
| 6-7 | return code | Return code from the JBOD operation |
| 8-263 | block | Where needed, a block of size JBOD_BLOCK_SIZE |

Table 1: JBOD procotol packet format

## Implementation

In addition to replacing all calls in `mdadm.c`, you will implement functions defined in `net.h` in the provided `net.c` file. Specifically, you will implement `jbod_connect` function, which will connect to JBOD_SERVER at port JBOD_PORT, both defined in `net.h`, and `jbod_disconnect` function, which will close the connection to the JBOD server. Both of these functions will be called by the tester. The file `net.c` contains some functions with empty bodies that can help with structuring your code, but you don't have to implement them. Other than implementing the functions in `net.h`, it is up to you how you structure your code.

## Testing

Once you finish implementing your code, you can test it by running the provided `jbod_server` in one terminal, which implements the server component of the protocol, and running the `tester` with the workload file in another terminal. Below is a sample session from the server and the client:

Output from the `jbod_server` terminal:

```
$ ./jbod_server
JBOD server listening on port 3333...
new client connection from 127.0.0.1 port 32402
client closed connection
```

Output from the `tester` terminal:

```
$ ./tester -w traces/simple-input >my-out
connected to the JBOD server at 127.0.0.1
Cost: 0
Hit rate:  -nan%
closed connection to the JBOD server
```

You can also run the `jbod_server` in verbose mode to print out every command that it receives from the client. Below is sample output that was trimmed to fit the space.

```
$ ./jbod_server -v
JBOD server listening on port 3333...
new client connection from 127.0.0.1 port 38546
received cmd id = 0 (JBOD_MOUNT) [disk id = 0 block id = 0], result = 0
received cmd id = 2 (JBOD_SEEK_TO_DISK) [disk id = 0 block id = 0], result = 0
received cmd id = 5 (JBOD_WRITE_BLOCK) [disk id = 0 block id = 0], result = 0
block contents:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

## Grading

There are not tests for this implementation. All of your 10 points will come from traces.