Assignment #4 – Network Attached Storage
CMPSC 311, Introduction to Systems Programming
Fall 2018
Due: April 29th 2018

The last assignment will extend the device driver you implemented in the previous assignment #3. You will extend your device driver to communicate with the HDD over a network. You will need to modify code from your previous assignment. While a superficial reading of this assignment made lead you to believe ***that this assignment work will be easy, it is not***. Please give yourself ample time to complete the code and debugging. Note that debugging this assignment will require you to debug your program interacting with a server program executing in another terminal window, so it can take some time.

Overview

This assignment is separated into two programs: the **client program** which you will complete the code for and the **server program** which is provided to you. To run the program, you will run the provided server executable in one terminal window and your compiled client executable in another. You will just use a local network connection to connect the two programs running on your machine (i.e., your computer will be both the client and the server—you're not actually communicating over the internet or between different computers though it's theoretically possible with minor tweaks to this project).

The challenge of this assignment is that you will be sending all parameters of hdd_data_lane over a network socket (along with receiving responses through that same socket). In fact, you'll be **replacing hdd_data_lane with your own function that sends the hdd_data_lane parameters to and from the server**. You must start with your hdd_file_io.c code from assignment #3. The assignment requires you to perform the following steps (after getting the starter code and copying over needed files from your implementation of the previous assignment):

1. Slightly modify your hdd_file_io.c to remove 3 functions

Before delving into the networking code, this assignment simplifies (in a sense) the API you've been using to communicate with the device. Instead of having all of the following functions:

- hdd_data_lane
- hdd_read_block_size
- hdd_delete_block
- hdd_initialize

We'll now just have one:

- hdd_data_lane

To remove the three old functions, just make the following replacements:

- int hdd_initialize() becomes:
    - HddBitResp hdd_data_lane(HddBitCmd cmd, void * data) where

        - HddBitCmd cmd = all 0s except the op field is HDD_DEVICE and the flags field is HDD_INIT
        - void * data = NULL
        - Just check HddBitResp's R bit to now determine success or failure of initialization
- hdd_delete_block becomes:
    - HddBitResp hdd_data_lane(HddBitCmd cmd, void * data) where
        - [When deleting any block except the meta block] HddBitCmd cmd = all 0s except the op field is HDD_DELETE and the block ID field is the block you'd like to delete
        - [When deleting the meta block] HddBitCmd cmd = all 0s except the op field is HDD_DELETE and the flags field is HDD_META_BLOCK
        - void * data = NULL
        - Just check HddBitResp's R bit for success or failure
- hdd_read_block_size becomes
    - Nothing. You can finally save the size of a block.

Note your other hdd_data_lane calls don't need changing unless you've been manually typing in 3 for your op code instead of HDD_DEVICE. If you have been, simply replace all occurrences of 3 with HDD_DEVICE.

2. Minor naming change in hdd_file_io.c and adding an include

We're now going to replace all hdd_data_lane function calls with the name hdd_client_operation. The hdd_client_operation function is defined in a new file hdd_client.c as follows:

HddBitResp hdd_client_operation(HddBitCmd cmd, void * data);

You can see it accepts and returns the same values as hdd_data_lane. Abstractly, this function performs the same function as the original hdd_data_lane function it is replacing. What it does specifically will be explained in the next section. For now, just make the name change and include hdd_network.h in your hdd_file_io.c file.

3. Bulk of the project: hdd_client_operation

The remainder of the assignment is your implementation of the hdd_client_operation function in the **hdd_client.c file**. The idea is that you will coordinate with a server via a network protocol to transfer the commands and data from the client to the server. The server will execute the commands and modify the HDD storage accordingly. Thus, the client code you're writing is just a messenger for what used to be hdd_data_lane commands.

The first time you want to send a message to the server, you have to connect. Connect to address HDD_DEFAULT_IP and port HDD_DEFAULT_PORT, both of which are defined in the **hdd_network.h** file.

To transfer the the HddBitCmd commands, you send the 64-bit request values over the network and receive the 64-bit response values. Note that you need to convert the 64 bit-values into **network byte order before sending** them and converting them to **host byte order when receiving them**. The functions htonll64 and ntohll64 are used to perform these conversions respectively and are provided for you in the cmpsc311_util.h file.

Note that extra data needs to be sent or received for certain HddBitCmd's (i.e. when reading from a block or writing to a block), but not for all of them. See **Table 1** for information about which requests should send and receive buffers.

Op Field of HddBitCmd	Flags Field of HddBitCmd	What You Send	What You Receive
HDD_DEVICE	HDD_INIT	HddBitCmd (only)	HddBitResp (only)
	HDD_FORMAT		
	HDD_SAVE_AND_CLOSE		
HDD_BLOCK_CREATE	HDD_NULL_FLAG	HddBitCmd and bytes of block	HddBitResp (only)
	HDD_META_BLOCK		
HDD_BLOCK_OVERWRITE	HDD_NULL_FLAG	HddBitCmd and bytes of block	HddBitResp (only)
	HDD_META_BLOCK		
HDD_BLOCK_READ	HDD_NULL_FLAG	HddBitCmd (only)	HddBitResp and bytes of block
	HDD_META_BLOCK		
HDD_BLOCK_DELETE	HDD_NULL_FLAG	HddBitCmd (only)	HddBitResp (only)
	HDD_META_BLOCK		

Table 1: HDD request messages: data sent and received with each HddBitCmd and flag pair

On sends that require sending a buffer, you simply send the buffer immediately after the 64-bit value. On receives, you first receive the 64-bit response value, convert it to host byte order, and extract the op type and block size. If the command requires a buffer be received, then you should receive data of length equal to that returned in the HddBitResp's block size.

Note that the last thing you should do in your client program is after a HDD_SAVE_AND_CLOSE op type and is to disconnect from the server (i.e., close the socket).

To assist with connecting to the network and sending/receiving data, see the network **slides posted on Canvas under assignment #4**. Read them all, but specifically see Slide 34 for an overview and slide 49 for sample code demonstrating how you'd connect and send/receive bytes.

Testing

[Note: please run "sudo apt-get install libgcrypt20-dev" before testing. Probably won't matter for most of you but it might for a few others]

Overall, the testing is very similar to the previous assignment. One of the differences is that you'll need to open **two windows and execute the client in one and the server in the other**. To run the server, use the command:

./hdd_server –v

The server will run continuously without being restarted (unless you send the server something unexpected that causes it to halt). That is, the client can run several workloads by connecting to the server and sending commands. To run the client, use the command:

./hdd_client -v <test arguments>

The first phase of testing the program is performed by using the unit test function for the

hdd_file_io interface. The main function provided to you simply calls the function hddIOUnitTest. As before it's with the following flags:

./hdd_client -u –v

If the program completes successfully, the following should be displayed as the last log entry:

HDD unit tests completed successfully.

The second phase of testing will run workloads just like before, but now there's three. These will test the mounting and unmounting of just like before and will save the state of the block storage to hdd_conent.svd. Make sure you run these commands sequentially as seen below:

./hdd_client -v workload-one.txt
./hdd_client -v workload-two.txt
./hdd_client -v workload-three.txt

If the program completes successfully, the following should be displayed as the last log entry for each workload:

HDD simulation completed successfully.

The last phase of testing will extract the saved files from the filesystem. To do this, you will use the –x option:

./hdd_client -v -x simple.txt

This should extract the file simple.txt from the device and write it to your current directory. Next, use the diff command to compare the contents of the file with the original version simple.txt.orig distributed with the original code:

diff simple.txt simple.txt.orig

If they are identical, diff will give no output (i.e, no differences). **Repeat these commands to extract and compare the content for the files raven.txt, hamlet.txt, penn-state-alma-mater.txt, firecracker.txt, and solitude.txt, and o44.txt (note the o44.txt is new)**. Just like before don't try to extract the same file twice without first deleting the newly created .txt file.

Note: Like all assignments in this class you are prohibited from copying any content from the Internet or discussing, sharing ideas, code, configuration, text, or anything else or getting help from anyone in or outside of the class. Consulting online sources is acceptable, but under no circumstances should *anything* be copied. Failure to abide by this requirement will result dismissal from the class as described in our course syllabus.
