Assignment #5 - Network Attached Storage

CMPSC311 - Introduction to Systems Programming
Fall 2014 - Prof. McDaniel

Due date: December 12, 2014 (11:59pm)

The next assignment will extend the device driver you implemented in the previous assignment. You will extend your device driver to communicate with the CRUD device 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 the 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 an server program executing in another window, so it can take some time.

Overview

For this assignment, the operation of the CRUD device is separated into two programs; the `crud_client` program which you will complete the code for, and the `crud_server` program which is provided to you. To run the program, you will run the server in one window and the client in another. You will use a local network connection (called the loopback interface) to connect the two programs running on the same machine (yours).

The challenge of this assignment is that you will be sending your CRUD requests and receiving responses over a network. You **must start with your code from assignment #4**, as it requires it. 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 assignments):

- The first thing you are to do is to replace the all of the calls to `crud_bus_request` in your code to:

  ```
  CrudResponse crud_client_operation(CrudRequest op, void *buf);
  ```

  This function is defined in the new file `crud_network.h` and performs the same function as the original bus request function it is replacing. The assignment should not require any other changes to your code from assignment #4.

- The remainder of the assignment is your implementation of the `crud_client_operation` function in the `crud_client.c` code file. The idea is that you will coordinate with the a server via a network protocol to transfer the commands and data from the client memory to the server memory. The server will execute the commmands and modify the CRUD storage accordingly.

The first time you want to send a message to the server, you have to connect. Connect to address `CRUD_DEFAULT_IP` and port `CRUD_DEFAULT_PORT`. To transfer the 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 functions respectively.

Note that extra data needs to be sent or received for certain CRUD requests, but not for all of them. See Table 1 for information about which requests should send and receive buffers. 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 command and length. If the command requires a buffer be received, then you should receive data of length equal to that returned in the response. For more detail, see the pseudocode provided in the assignment presentation made in class (which is available on the course website).

You will use the network APIs described in the associated lecture to complete this assignment. Note that the last thing you should do in your client program after a `CRUD_CLOSE` is to disconnect.

Instructions

You are to build the enhanced version of the user-space device driver. To do this, you should perform the following steps in completing the assignment:

Command	Send	Receive
CRUD_INIT	CrudRequest value (only)	CrudResponse value (only)
CRUD_FORMAT	CrudRequest value (only)	CrudResponse value (only)
CRUD_CREATE	CrudRequest value, data of size length specified in request field	CrudResponse value (only)
CRUD_READ	CrudRequest value (only)	CrudResponse value, data of size length specified in response field
CRUD_UPDATE	CrudRequest value, data of size length specified in request field	CrudResponse value (only)
CRUD_DELETE	CrudRequest value (only)	CrudResponse value (only)
CRUD_CLOSE	CrudRequest value (only)	CrudResponse value (only)

Table 1: CRUD request messages: data sent and received with each CRUD command

1. From your virtual machine, download the starter source code provided for this assignment. To do this, use the `wget` utility to download the file off the main course website:

   `http://www.cse.psu.edu/~mcdaniel/cmpsc311-f14/docs/assign5-starter.tgz`

2. Change to the directory you are using for your assignments and copy the file into it.

```
% cd ~/cmpsc311
% cp assign5-starter.tgz cmpsc311
% cd cmpsc311
% tar xvfz assign5-starter.tgz
```

3. Copy the file `crud_file_io.c` from assignment #4 into the `assign5` directory. Note that **you must use your code from the previous assignment**. No substitutions with other people's code.

4. Implement `crud_client_operation`, as described in the previous section, and use it to replace all calls to `crud_bus_request`.

Testing

- The key to testing is to open two windows and execute the client in one and the server in the other. To run the server, use the command:

```
% ./crud_server -v
```

  The server will run continuously without being restarted. That is, the client can run several workloads by connecting to the server and sending commands. To run the client, use the command:

```
% ./crud_client -v <test arguments>
```

- The first phase of testing the program is performed by using the unit test function for the `crud_file_io` interface. The main function provided to you simply calls the function `crudIOUnitTest`. If you have implemented your interface correctly, it should run to completion successfully. To test the program, you execute the simulated filesystem using the `-u` and `-v` options:

```
./crud_client -u -v
```

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

```
CRUD unit tests completed successfully.
```

- The second phase of testing will run two workloads on your filesystem implementation. To do this, run the following commands on the

```
./crud_client -v workload-one.txt
./crud_client -v workload-two.txt
./crud_client -v workload-three.txt
```

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

```
CRUD 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:

```
./crud_client -v -x simple.txt
```

  This should extract the file `simple.txt` from the device and write it to the local filesystem. 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`

To turn in:

1. Create a tarball file containing the `assign5` directory, source code and build files. Email the program to `mcdaniel@cse.psu.edu` and the section TA by the assignment deadline (11:59pm of the day of the assignment). The tarball should be named `LASTNAME-PSUEMAILID-assign5.tgz`, where LASTNAME is your last name in all capital letters and PSUEMAILID is your PSU email address without the ”@psu.edu”. For example, the professor was submitting a homework, he would call the file `MCDANIEL-pdm12-assign5.tgz`. **Any file that is incorrectly named, has the incorrect directory structure, or has misnamed files, will be assessed a one day late penalty.**

2. Any incorrect tarball (containing something other than your completed code for this assignment) will be considered as not being submitted at all. We will try to notify you if we notice something is amiss, but it is up to you to confirm that the tarball is good.

3. Before sending the tarball, test it using the following commands (in a temporary directory – NOT the directory you used to develop the code):

```
% tar xvzf LASTNAME-PSUEMAILID-assign5.tgz
% cd assign5
% make
... (TEST THE PROGRAM)
```

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.

Honors Option

Continue implementing the device driver with the additional restriction that each object in the object store can be no longer than 1024 bytes. This requires the system to track multiple ordered objects on the storage device.

In addition, you must modify the client program code (in `crud_sim`) to add a new feature. This feature will copy a file into the CRUD storage device using a `-c <file>` input parameter. To do this, it should mount the CRUD device and send consecutive writes of one kilobyte or less of the file content into the CRUD device. Once done, you should unmount the device. You should test the program by sending in several files and extracting them.