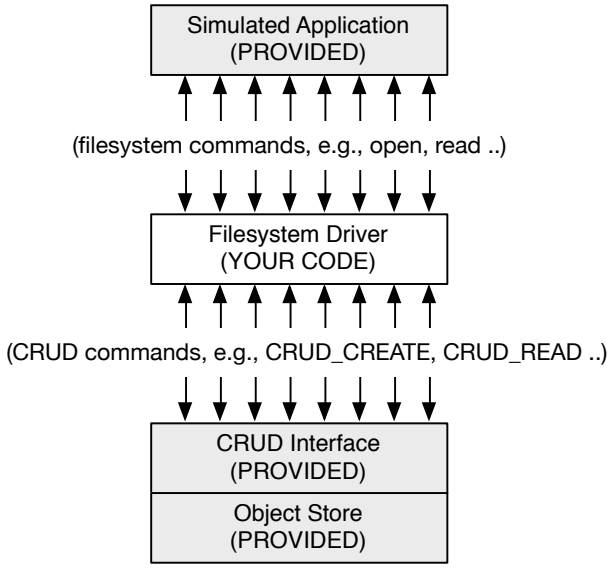
Assignment #3 - CRUD Device Driver (Ver 1.0)
CMPSC311 - Introduction to Systems Programming
Fall 2014 - Prof. McDaniel

Due date: October 13, 2014 (11:59pm)

All remaining assignments for this class are based on the creation and extension of a user-space device driver for a filesystem built on top of a object storage device. At the highest level, you will translate file system commands into storage array commands. These file system commands include open, read, write, and close for files that are written to your file system driver. These operations perform the same as the normal UNIX I/O operations, with the caveat that they direct file contents to the object storage device instead of the host filesystem. The arrangement of software is as follows:



Object Store/CRUD Interface

An object store is a virtual device that stores variable sized blocks of data called objects. Software that uses the store create and manipulate objects on the device in a way very similar to normal disk drives (see the CRUD interface commands below), except that they manipulate objects instead of disk blocks. Each object is referenced by a uniquely identified by an integer value assigned by the object store called an object identifier.

The object store you are building on top of exports a *CRUD* interface; it supports **c**reating objects, **r**eading objects, **u**pdating objects, and **d**eleting objects. Note that this code for this interface is provided to you in library form. Also, objects have *immutable* size; once allocated, the contents can be changed repeatedly, but the size can never change. Thus, any operation which would require a change to the object size must be performed by deleting an old object and creating a new one.

You are to use the CRUD interface to make requests to the object store, as defined in the file `crud_driver.h`,. This interface contains a single function call that accepts two arguments, a 64-bit CRUD bus request value (`with type CrudReuqest`) and a pointer to a variable-sized buffer;

```
CrudResponse crud_bus_request(CrudRequest request, void *buf);
```

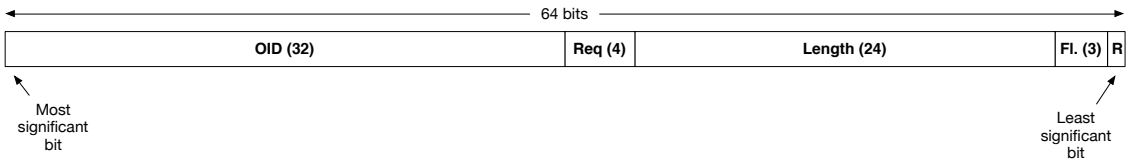
The CRUD commands are as follows:

- `CRUD_INIT` - This command initializes the object store and readies it for use in file operations. This request MUST be called before any others.
- `CRUD_CREATE` - This command creates an object whose length is defined in a buffer request value defined below. The buffer passed to the function should contain data of that length. Note that the CRUD interface

copies the passed data into an internal structure, so you are responsible for managing any buffers passed to it. If successful, the operation will return the new object ID in the response object value.

- `CRUD_READ` - This command reads an object (in its entirety) from the object store and copies its contents in passed buffer. The length field should be set to the length of the passed buffer (because you don't know how big the object is going to be, you should always pass in a buffer of size `CRUD_MAX_OBJECT_SIZE`). The returned response value indicates the length of the object read.
- `CRUD_UPDATE` - This command will update the contents of an object. Note that the object size CAN NEVER change. Thus, the call will fail unless the buffer sent is the same size as the original object created.
- `CRUD_DELETE` - This command deletes the object, making it unavailable for later access. Note that the object ID of a object may be reused later.

The CrudRequest value passed to the bus function is a 64-bit value that defines a request and its parameters, as follows (by convention bit zero is the most significant bit):



Note that the 64-bit response value (`CrudResponse`) returned from a call to the CRUD interface has the same fields, with slightly different meaning as described below. The fields of the value are:

- OID - This is the object identifier of the object you are executing a command on. For object creates, the returned object identifier is the new object ID.
- Req - This is the request type of the command you are trying to execute. The value can be `CRUD_INIT`, `CRUD_CREATE`, `CRUD_READ`, `CRUD_UPDATE`, or `CRUD_DELETE`.
- Length - This is the length of the object. On updates and creates, the request value should include the size of the object you are attempting to create or update. On reads, the length should be the size of the buffer you are handing to the device (the maximum size of the object you can read). For all calls, the returned size is the size of the object read, written, or updated.
- Flags - These are unused for this assignment.
- R - Result code. Only used in the response, this is success status of the command execution, where 0 (zero) signifies success, and 1 signifies failure. You must check the success value for each bus operation.

The Filesystem Driver

The bulk of this assignment is to develop code for the file-oriented input/output driver that uses the object store. Conceptually, you are to translate each of the below file I/O function calls into calls to the previously described CRUD interface. It is up to you to decide how to implement these functions. However, the functions must maintain the file contents in exactly the same way as a normal filesystem would. The functions that you are to implement are defined in `crud_file_io.h` and `crud_file_io.c` and should perform as follows:

Function	Description
crud_open	This call opens a file and returns an integer file handle (to be assigned by you). For this assignment, the file is always assumed to be empty (zero length) when opened.
crud_close	This call closes the file referenced by the file handle. For this assignment, you are to delete all of the contents associated with the file when it is closed.
crud_read	This call reads a *count* number of bytes from the current position in the file and places them into the buffer *buf*. The function returns -1 if failure, or the number of bytes read if successful. If there are not enough bytes fulfill the read request, it should only read as many as are available and return the number of bytes read.
crud_write	This call writes a *count* number of bytes at the current position in the file associated with the file handle *fd* from the buffer *buf*. The function returns -1 if failure, or the number of written read if successful. When number of bytes to written extends beyond the end of the file, the size of the file is increased.
crud_seek	This call resets the current position of the file associated with the file handle *fd* to the position *loc*.

The central constraint to be enforced on your code is that it can only maintain meta-information about an open file. Put more directly, your code can not maintain any file content or length information in local structures–all such data must be maintained by the object store. Also note that for the purposes of this assignment, you can make the following assumptions:

- No file will become larger than the maximim object size (CRUD_MAX_OBJECT_SIZE).
- Your program will never have more than one file open at a time.

Instructions

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

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/assign3-starter.tgz

2. Change to your assignments a directory for your assignments and copy the file into it.

```
% cd  /cmpsc311
% cp assign3-starter.tgz cmpsc311
% cd cmpsc311
% tar xvfz assign3-starter.tgz
```

3. Complete the code for the file I/O functions defined in crud_file_io.c as described above. You should be able to use the Makefile provided to build the program without modification.

4. Add comments to all of your files stating what the code is doing. Augment the comment function header for each function you are defining in the code.

5. The testing of 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, as:

   ./crud_sim -u -v

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

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

To turn in:

1. Create a tarball file containing the `assign3` 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-assign3.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-assign3.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. 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-assign3.tgz
% cd assign3
% 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

Place 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.