Assignment #4 - The CRUD Filesystem Driver

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

Due date: November 17, 2014 (11:59pm)

Overview

In this assignment, you will extend the device driver you implemented in Assignment #3. To be clear: **you must start this assignment using the code that *you* submitted for Assignment #3**. You will be implementing several new features in your driver which add support for some important operations of real-world filesystems. When you are done, your driver will support:

- Tracking multiple open files simultaneously, by using a provided file table
- File data that persists between runs of a program, by implementing the basic filesystem operations of "format," "mount," and "unmount"

To help support your implementation, several new features (described in detail in the next section) are provided for you by the CRUD interface and `libcrud.a`:

- Definitions of a file allocation table structure and file table to use in your code
- A special "priority object" in the object store which can be accessed at any time
- New `CRUD_FORMAT` and `CRUD_CLOSE` operations and an enhanced `CRUD_INIT` command to support the format, mount, and unmount operations

New CRUD Features

This section describes the new features of the CRUD object store which are *provided for you by the CRUD library!* You do not need to implement them yourself, but you will be using them to implement your portion of the assignment. All other parts of the CRUD interface behave exactly as they did in the previous assignment.

File Allocation Table

The most important new feature is the definition of a "file allocation table" for use in your implementation. Each entry in this table represents a single file in the object store and its current status if open. The structure of a file table entry is defined as `CrudFileAllocationType` in `crud_file_io.h`:

```
typedef struct {
    char      filename[CRUD_MAX_PATH_LENGTH];
    CrudOID   object_id;
    uint32_t  position;
    uint32_t  length;
    uint8_t   open;
} CrudFileAllocationType;
```

and the fields are defined as follows:

- `filename`: Name of the file, as passed to `crud_open`; you may assume that this name, including the terminator, will never be longer than `CRUD_MAX_PATH_LENGTH`
- `object_id`: OID of the object which corresponds to this file
- `position`: Current file position (only for open files)
- `length`: Length of the file in bytes

- `open`: Flag that indicates whether the file is currently open (nonzero value) or closed (zero)

The file table (array) itself is declared in `crud_file_io.c`:

```
CrudFileAllocationType crud_file_table[CRUD_MAX_TOTAL_FILES];
```

This data structure serves two purposes. First, it can be used while the program is running to maintain information about the state of any open files. Second, it can be stored in the object store when the filesystem is unmounted, then loaded again at the next mount, making file information persist across multiple successful runs of the program.

Priority Object

The other new feature of the object store is the addition of a "priority object." This is a single, special object which is remembered by the object store, and it can be manipulated *without needing to know its OID*. This will be useful when storing the persistent file allocation table in the object store. To create, read, update, or delete the priority object, issue the corresponding CRUD request with an OID of 0 and the flags set to `CRUD_PRIORITY_OBJECT` (defined in `crud_driver.h`).

INIT, FORMAT, and CLOSE Commands

The CRUD command set has been augmented to allow object contents to be made persistent by saving the data to a file called `crud_content.crd`. You do not need to create this file or write to it yourself. Instead, the object store will manipulate the content file when you issue the following requests:

- `CRUD_INIT`: In addition to initializing the object store, this command will now also load any saved state from `crud_content.crd` (if that file exists). As in the previous assignment, you must issue an INIT request exactly once, before any other command is sent to the CRUD hardware.
- `CRUD_FORMAT`: This new command will delete `crud_content.crd` and all objects in the object store (including the priority object).
- `CRUD_CLOSE`: This new command will save the contents of the object store to the `crud_content.crd` file, creating it if it does not exist.

These new commands will be useful when implementing the new format, mount, and unmount features.

Getting the Code

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

`http://www.cse.psu.edu/~mcdaniel/cmpsc311-f14/docs/assign4-starter.tgz`

After you have downloaded the tarball, copy it into the directory you have been using for your assignments. Change to that directory and untar the file.

```
% cp assign4-starter.tgz ~/cmpsc311
% cd ~/cmpsc311
% tar -xvzf assign4-starter.tgz
```

Procedure

To implement the enhanced version of the user-space device driver, perform the following steps:

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

2. Modify your current functions so that they use the `crud_file_table` to keep track of information open files, instead of your own table or variables.

3. Implement support for multiple open files by using the file allocation table. To do this, whenever a new (not existing) file is opened, assign it an unused slot in the table. Copy the filename and set the initial contents to empty. From this assignment forward, *closing a file should no longer delete its contents*; instead, it should just clear the open flag (i.e., set it to zero). Then, when opening an existing file, you reset the position to zero and set the open flag again.

   The file handle which is returned from `crud_open` should be the file's index in the file allocation table. This will allow you to find the right entry easily in the other filesystem functions.

4. Implement three new functions to handle operations involving the file allocation table:

   (a) `crud_format`: Reinitializes the filesystem and creates an empty file allocation table. This function should perform a normal `CRUD_INIT` followed by a `CRUD_FORMAT`, initialize the file allocation table with zeros (signifying that all slots are unused), and save the table by creating a priority object containing the table data.

   (b) `crud_mount`: Loads an existing saved filesystem into the object store and file table. This function should first perform a normal `CRUD_INIT` if it has not already been run. Then it should locate the file allocation table (by reading the priority object) and copy its contents into the `crud_file_table` structure.

   (c) `crud_unmount`: Saves all changes to the filesystem and object store. This function should store the current file table back in the storage device by updating the priority object. Once that is done, it should issue a `CRUD_CLOSE` request to the device, which will write out the persistent state file and shut down the virtual hardware.

Testing

- 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_sim -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_sim -v workload-one.txt
  ./crud_sim -v workload-two.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_sim -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`.

Submission

1. Create a tarball file containing the `assign4` 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 the assignment is due). The tarball should be named `LASTNAME-PSUEMAILID-assign4.tgz`, where LASTNAME is your last name in all capital letters and PSUEMAILID is your PSU email address without the "@psu.edu". For example, if the professor were submitting a homework, he would call the file `MCDANIEL-pdm12-assign4.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 verify that the tarball is correct.

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

```
% tar -xvzf LASTNAME-PSUEMAILID-assign4.tgz
% cd assign4
% make clean
% 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. Note that you will likely have to modify the persistence functions.