Assignment #3 - LionCloud (v 1.1)

Due Apr 17 by 11:59pm **Points** 125 **Submitting** a file upload
File Types tgz and tar.gz **Available** Mar 21 at 12am - Apr 20 at 11:59pm about 1 month

This assignment was locked Apr 20 at 11:59pm.

This next assignment will build upon your completed code for the previous assignment. You MUST use the the code turned in for the previous assignment. If you have bugs or incomplete code you have to fix it first. You are to implement new features and integrate code for the new features and workload. Unless stated otherwise, the architecture and behavior of the new system is the same as in the previous assignment.

Note: This assignment description provides an overview of the assignment. The associated slides (and in class presentation) provide more information, hints, and guidance for the project. Please refer to those materials as well in completing this assignment.

Feature Overview

You are to extend your device driver that sits between the virtual application and virtualized hardware devices. As before, the application makes use of the abstraction you provide called the LionCloud driver. In this next assignment, you will make modifications to the code to implement the following features:

1. The new manifest has more than one device.
2. The new workload has more than one file (in fact it has many). Reads may also span an arbitrary number of blocks, and be performed anywhere in the file.
3. There is a new operation that you must implement. This op code initializes the device and gets the number of blocks and sectors (which may be different for each device). The op code is called `LC_DEVINIT`. The frame values for `b0`, `b1`, and `c0` as the same as in the previous assignment. The other frame values are:

Operation	Description	Remaining register use
LC_DEVINIT	Initialize interface to a specific device	When sending: ◦ c1 - device to initialize ◦ c2, d1, d0 - 0 When receiving ◦ c1 - device initialized ◦ c2 - 0 ◦ d0 - the number of sectors in this device ◦ d1 - the number of blocks per/sectopr in this device

You **cannot** depend on the constant size of the device info:

```
#define LC_DEVICE_NUMBER_SECTORS 10
#define LC_DEVICE_NUMBER_BLOCKS 64
```

These numbers are not even defined any more. The idea is that once you did the LC_PROBE and find that a device exists, you will have to LC_DEVINIT and learn how many sectors and blocks there are for each device. You then have to dynamically allocate memory to hold the information you had for your previous device.

4. Implement a LRU cache that caches blocks retrieved and written to the LionCloud devices. The API for the cache is provided in lc_cache.h and the starter code for the implementation is in lc_cache.c. You will need to:
   - Implement all of the functions defined in the file.
   - Add code to call the functions from your driver implementation. In particular, any place you get a block from the driver, you should check first if it is in the cache.
   - You should collect statistics on how many hits and misses you have. Print out the values and your hit ratio when you close the cache.

   The cache should be fully associative (any block can go in any cache line). It should be dynamically allocated when `lcloud_initcache` is called. Each line should contain the device/sector/block and data. You can use a hash table or perform a linear search over this data structure to find entries or empty lines. You may also use other data structures as you see fit.

   It is recommended to implement LRU by tracking when each element was last used, and linearly searching for the oldest element as a candidate for eviction.

Honors Option

Create an independent LRU cache for each device in the system. You should not change the API, but hide the details of the caches within your implementation. You should provide a nice looking output showing the performance of each cache as well as the total performance when you close the cache.

Instructions

1. Login to your virtual machine. From your virtual machine, download the starter source code provided for this assignment. To do this, go to the canvas website and download the file assign3-starter.tgz (you should be able to click the link **here**).
2. Create a directory for your assignments and copy the file into it. Change into that directory.

```
% mkdir cmpsc311
% cp assign3-starter.tgz cmpsc311
% cd cmpsc311
```

```
% tar xvzf assign3-starter.tgz
% cd assign3
```

Once unpacked, you will have the starter files in the assign3, including the source code, libraries, some workload and manifest files and a Makefile. Note that there is a file **cmpsc311-assign3-sample-output.txt** that contains sample output from my implementation of the assignment.

3. Copy your code from the previous assignment. For most the only file you will need is the `lcloud_filesys.c` file. To copy, simply use the UNIX `cp` command. For example:

   `cp ../assign2/lcloud_filesys.c .`

   NOTE: be sure to make a copy of the original file before you start. A good way to do this is to email yourself a copy of the code before you get started.

4. You are to complete the `lcloud_cache.c` functions defined above, as well as extend the functionality in the `lcloud_filesys.c` to meet the requirements above. Note that you may need to create additional supporting functions within the same file. Include functional prototypes for these functions at the top of the file.

5. Add comments to all of your files stating what the code is doing. Fill out the comment function header for each function you are defining in the code. A sample header you can copy for this purpose is provided for the main function in the code.

6. To test your program with these provided workload files, run the code specifying a workload file as follows:

   `./lcloud_sim -v cmpsc311-assign3-manifest.txt cmpsc311-assign3-workload.txt`

   Note that you don't necessarily have to use the -v option, but it provides a lot of debugging information that is helpful. If the program completes successfully, the following should be displayed as the last log entry:

   LionCloud simulation completed successfully!!!

To turn in:

1. Create a tarball file containing the `assign3` directory, source code and build files as completed above. Submit the final tar file through canvas by the assignment deadline. 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 ~ see above)
```

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.

Rubric

Criteria	Ratings			Pts
Reading/writing to multiple devices supported.	**15.0 pts** **Full Marks**	**7.5 pts** **Half**	**0.0 pts** **No Marks**	15.0 pts
Reading/writing to multiple files supported.	**15.0 pts** **Full Marks**	**7.5 pts** **Half**	**0.0 pts** **No Marks**	15.0 pts
LC_DEVINT implemented correctly to get respective sectors/blocks per device.	**15.0 pts** **Full Marks**	**7.5 pts** **Half**	**0.0 pts** **No Marks**	15.0 pts
Init/close cache implemented correctly.	**15.0 pts** **Full Marks**	**7.5 pts** **Half**	**0.0 pts** **No Marks**	15.0 pts
Get/put cache implemented correctly.	**15.0 pts** **Full Marks**	**7.5 pts** **Half**	**0.0 pts** **No Marks**	15.0 pts
Driver finishes successfully.	**25.0 pts** **Full Marks**	**12.5 pts** **Half**	**0.0 pts** **No Marks**	25.0 pts
All functions documented well.	**12.5 pts** **Full Marks**	**6.25 pts** **Half**	**0.0 pts** **No Marks**	12.5 pts
Compiles successfully (must have at least one of the other sections completed to receive credit).	**12.5 pts** **Full Marks**	**6.25 pts** **Half**	**0.0 pts** **No Marks**	12.5 pts
Formatting deduction	**0.0 pts** **Full Marks**		**0.0 pts** **No Marks**	0.0 pts
Note: Partial credit given based on where validation fails.	**0.0 pts** **Full Marks**		**0.0 pts** **No Marks**	0.0 pts
				Total Points: 125.0