# Assignment 4 - Block Cache

**Due** Aug 7, 2019 by 11:59pm **Points** 150 **Submitting** a file upload
**File Types** tar.gz, tar.gz, tgz, and tgz
**Available** Jul 23, 2019 at 11:59pm - Aug 10, 2019 at 11:59pm 18 days

This assignment was locked Aug 10, 2019 at 11:59pm.

# Assignment #4 - Block Cache
# CMPSC311 - Introduction to Systems Programming
# Summer 2019 - Prof. McDaniel

**Due date:** August 5, 2019 (11:59pm)

## Overview

In this assignment you will again extend the driver written in the previous assignments. Everything about the block device remains as before, except as described in the specification here. At the highest level, you will extend the code to support many and larger files and frame caching. All of the extensions will be made to the functions modified in the previous section as well as a new file `block_cache.c`.

## Block Driver Extension

The code addition to the driver will be the ability to support multiple open files at a given time, as well as a frame cache for your block device. More specifically, the driver must be able to support `BLOCK_MAX_TOTAL_FILES` files, all of which may be open at the same time. The files have no maximum size. And secondly, you will also add a frame cache to your driver implementation. It will store dynamically allocated frames in a **write through LRU cache** you will implement. The function declarations are provided to you in `block_cache.h` and shown below. You will need to check the cache every time you read a frame to see if is already there and used the cached entry rather than going to the bus. On writes, you will need to insert it into the cache if it doesn't exist or update it if it does exist. The cache should allow for a maximum number of entries (the cache size). When the number of entries is exceeded, you need to delete a frame (and deallocate the frame) per the cache replacement policy. The performance of your cache will be evaluated with nine different cache sizes: 5, 10, 25, 50, 100, 250, 500, 1000, and 1024 cache entries. You can specify a cache size by invoking `block_sim` with the '-c' option along with the size: `./block_sim -v -c 10 <workload_file>`. You will be graded based on the performance of your cache (with each of the listed cache sizes) against the workload.

Given the policy, the target performance to achieve for each size is:

| Cache size | Target hit ratio (%) |
|---|---|
| 5 | 52.19% |
| 10 | 53.71% |
| 25 | 58.34% |
| 50 | 67.34% |
| 100 | 88.02% |
| 250 | 99.12% |
| 500 | 99.12% |
| 1000 | 99.12% |
| 1024 | 99.12% |

The functions you must complete are as follows:

```
int set_block_cache_size(uint32_t max_frames);
// Set the size of the cache (must be called before init)

int init_block_cache(void);
// Initialize the cache

int close_block_cache(void);
// Clear all of the contents of the cache, cleanup

int put_block_cache(BlockIndex blk, BlockFrameIndex frm, void *frame);
// Put an object into the object cache, evicting other items as necessary

void * get_block_cache(BlockIndex blk, BlockFrameIndex blk);
// Get an object from the cache (and return it)
```

You should refer to the lecture on caching for hints on how to implement the cache. Note that you will have to insert calls to the above functions in the existing code, including during initialization of the driver, shut down, and during reads and writes. The `set_block_cache_size` function is called by the main program to set the maximum number of frames (and thus you don't have to call this in your code).

# Assignment Details

Below are the step by step details of how to implement, test, and submit your device drivers as part of the class assignment. As always, the instructor and TAs are available to answer questions and help you make progress. Note that this assignment is deceptively complicated and will likely take even the best students a substantial amount of time. Please plan to allocate your time accordingly.

1. From your virtual machine, download the starter source code provided for this assignment **here**.
2. Unpack the starter files **into a temporary directory** with the command:
   `tar xvfz assign4-starter.tgz`
3. Once unpacked, there will be some new files and some new versions of old files which we need to migrate over to your working directory. Before moving further, we advise to copy your assign3 directory over to a new directory for assign4. We will copy the following starter files into the **new** assign4 directory:
   `Makefile`
   `block_sim.c`
   `block_cache.c`
   `block.cache.h`
   `block_controller.h`
   `libblocklib.a`
   `workload/` (replace the old workload directory with this new one)

   **Important:** Copy **only** the files listed above into the new assign4 directory.
4. Your task is to implement the cache functionality as described above, adding any necessary changes to functions in `block_driver.c` to use your cache.
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, you will run it with the sample workload file provided. To do this, run the program from the command line with:

   ```
   make clean && make
   ```

   ```
   ./block_sim -v -c <cache_size> workload/cmpsc311-sum19-assign4-workload.txt
   ```

You must use the "-v" option before the workload filename when running the simulation to get meaningful output. Once you have your program running correctly, you should see the cache performance statistics along with following message at the end of the output:

[INFO] BLOCK simulation completed successfully.

**Note**: Placement when the cache is not full can be fully associative.
**Note**: BlockIndex refers to the index of the block (in the case of multiple blocks). In this assignment, there is only one block, so index should always be 0.

# Extra credit

Complete the cache unit test function with code that initializes the cache, creates 10,000 cache operations of random frame gets and puts, then closes the cache and completely frees all of the memory associated with the created entries. Note that the `[-u]` option on the command line will run this and the other unit tests. The test should check that the data retrieved from the cache is correct (the bytes sent in are the same ones you get out).

## To turn in:

1. Create a tarball file containing the `assign4` directory, source code and build files as completed above. Submit the tarball to canvas by the assignment deadline (11:59pm of the day of the assignment). 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 was submitting a homework, he would call the file `MCDANIEL-pdm12-assign4.tgz`.
   **Note:** 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 submitting 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-assign4.tgz
   % cd assign4
   % 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.

---