Assignment #4 - Device Driver Cache Optimization
CMPSC311 - Introduction to Systems Programming
Fall 2013 - Prof. McDaniel

Due date: November 18, 2013 (11:59pm)

In this assignment you will extend your device driver implementation to include a block cache. Please read the following instructions carefully and perform the requested steps as directed.

SMSA Cache Specification

You are to implement a cache of blocks with a least recently used cache replacement policy. The cache structure `SMSA_CACHE_LINE` (the definition of a cache line) is defined in the `smsa_cache.h`. The cache should implement a LRU policy using the `used` field in the cache line and the `gettimeofday()` function. All read accesses (and read-before-write accesses) should check the cache before sending a request to the SMSA. Obviously, new cache items should be added to the cache when any block is read. The cache should implement a write-through cache strategy.

To implement the cache, the following functions must be implemented in `smsa_cache.c`:

- `int smsa_init_cache( uint32_t lines );` - this function initializes the cache by creating a set of cache lines that is equal to the number of `lines` in the program using dynamic memory management. Note that these cache lines will be reused throughout the run of the program (you do not need to allocate a cache memory structure after the the initialization is complete).
- `int smsa_close_cache( void );` - this function closes down the cache and releases all of the cache lines. Zero the data before releasing it. Note that any buffers that are in the cache (inserted during a previous `smsa_put_cache_line` should be released as well.
- `unsigned char *smsa_get_cache_line( SMSA_DRUM_ID drm, SMSA_BLOCK_ID blk );` – this function looks in the cache for a cache entry and returns the the cache line if available. If it is not in the cache, NULL is returned.
- `int smsa_put_cache_line( SMSA_DRUM_ID drm, SMSA_BLOCK_ID blk, unsigned char *buf );` – this function places the buffer into the cache, thus taking ownership of the buffer. If there is no more room in the cache, the least recently used block should be released and the new buffer added.

Note that this version of the SMSA measures performance of the disk array. At the end of each run of a workload, the program prints out the cycle count used by the SMSA. For example, a typical cycle count output would be:

```
        Wed Oct 23 07:32:01 2013 [OUTPUT] Cycle count [33755910]
```

The same simulator program you used from last assignment will be used. The code provided reads a *workload* file of virtual read commands and calls your implementation to serve then. The program is called as follows:

```
USAGE: smsa [-h] [-u] [-v] [-l <logfile>]  [-c <sz>] <workload-file>

where:
    -h - help mode (display this message)
    -u - run the SMSA unit test
    -v - verbose output
    -l - write log messages to the filename <logfile>
    -c - set cache size to <sz> lines

    <workload-file> - file contain the workload to simulate
```

The program will print out a good deal of output if you use the `-v` verbose flag. Use this to debug your program. It is also encouraged to use the logging facility provided by the `logMessage` function listed in the `cmpsc311_log.h` file. See its use in the simulator code for examples. To redirect the output into a utility you can use to walk through the data, redirect the output to `less`.

To do:

1. From your virtual machine, download the starter source code provided for this assignment. To do this, download the starter code linked to the announcements section of the course webpage (front page). Decrypt, untar, and unzip the repository using the gpg and tar tools as you did for the previous assignment. (You will need to use the password in the related slides to decrypt the file).
2. You are to use your own `smsa_driver.c` from the previous assignment (copy it into the assign4 directory). I am including my code for instruction only (so you can fix any problems you had with your code). I will not grade any assignment that uses my code for the driver.
3. You are to implement 4 functions defined in the interface header file `smsa_cache.h` whose implementation is in `smsa_cache.c` defined above.
4. You are to integrate calls to the cache functions as needed in your driver code.
5. Note that you may need to define a number of other support functions to make clean code here. All of these functions should check the correctness of every parameter and log an error and return -1 if any value is illegal.
6. Run the program with the example workload files and confirm that they run to completion. Sample output will be provided to compare against your run.
7. Create a text file called performance.txt. Run your program using a range of cache sizes (at least 5 that range no less than 3 orders of magnitude) over the workload files and write down in table form the performance of each. Look at the numbers and describe why they perform the way they do.

To turn in:

1. Create a GPG encrypted tarball file containing the `assign4` directory, source code and build files as completed above. Email the program to `mcdaniel@cse.psu.edu` and the section TA (listed on course website) by the assignment deadline (11:59pm of the day of the assignment). The tarball should be named `LASTNAME-PSUEMAILID-assign4.tgz.gpg`, 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-assign4.tgz.gpg`.

   Use the same password you used to decrypt the original file to encrypt this file.
2. Before sending the tarball, test it using the following commands (in a temporary directory – NOT the directory you used to develop the code):

   ```
   % gpg LASTNAME-PSUEMAILID-assign4.tgz.gpg
   % tar xvzf LASTNAME-PSUEMAILID-assign4.tgz
   % cd assign4
   % make
   ... (TEST THE PROGRAM)
   ```
3. Send the tar file in an email with the Subject Line **CMPSC311 ASSIGNMENT #4** to the professor and your section TA, with yourself CCed. YOU ARE RESPONSIBLE FOR DOING THIS CORRECTLY. Lost emails, emails not sent to both the professor and TA, bad tar files, bad encryption are your fault and it will be treated as late assignments. There will be no exceptions.

Bonus Points: You are to implement a second cache replacement policy that implements an random replacement policy (which removes a random cache line). Rerun the performance test and discuss the relative performance of the RR vs LRU policy for various workloads and cache sizes in the performance text file.

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 in dismissal from the class as described in our course syllabus.