CMPSC 311 - Introduction to Systems Programming

Module: Assignment #3

TAGLINE Device Driver (v1.1)

Professor Patrick McDaniel
Fall 2015
Homework #3

- This is an extension of the previous where you add more code to the existing functions and process much more complex workloads.

- Initial extensions
  - There are now up to maxlines number of taglines
  - You will see multiple reads and writes per call

```c
int tagline_read(TagLineNumber tag, TagLineBlockNumber bnum, uint8_t blks, char *buf);
// Read a number of blocks from the tagline driver

int tagline_write(TagLineNumber tag, TagLineBlockNumber bnum, uint8_t blks, char *buf);
// Write a number of blocks from the tagline driver
```
Homework #3

• Each block will be stored twice
  ‣ A primary block which will be used for reads
  ‣ A backup block used for recovery

Note: You must make sure that the blocks are on different disks!
Homework #3

- Periodically a disk will “fail”, which causes the disk to be in a failed state. You will write code for:
  
  ```c
  int raid_disk_signal(void);
  // A disk has failed which needs to be recovered
  ```

- Recovery:
  
  - This will use the new raid command STATUS to figure out which disk failed.
  - Format the failed disk (FORMAT bus command)
  - Use copies of the lost blocks on other disks to reconstruct all of the lost data.
Recovery

- Use copies of the lost blocks on other disks to reconstruct all of the lost data.
CMPSC 311 - Introduction to Systems Programming  
Module: Assignment #2  
TAGLINE Device Driver  

Professor Patrick McDaniel  
Fall 2015
Homework #2

• Idea you are to maintain the correct file contents in an object stored on the TAGLINE device operating on top of a disk array.

• 3 things to know
  ‣ How the tagline works
  ‣ How file fixed sized block I/O works
  ‣ How to make the object stuff look like files
The Tagline Device Driver

TAGLINE “Application”

TAGLINE Operations

TAGLINE Device Driver

RAID Array Operations

RAID Array Firmware

DISK 1  DISK 2  ...  DISK N
Assignment Basics

• You are to write the file operation functions defined in the tagline_driver.c
  ‣ Translate then tagline operations into RAID bus operations
  ‣ Maintain, in the storage device, the tagline contents
    • You must insert data, read data, and maintain file information
  ‣ For this assignment:
    • All blocks are exactly TAGLINE_BLOCK_SIZE size
    • No tagline will be larger than MAX_TAGLINE_BLOCK_NUMBER
    • The RAID array will contain RAID_DISKS with RAID_DISKBLOCKS blocks each

• A simulation of the tagline abstraction is provided for you that will exercise all of your code.
The tagline abstraction

- A tagline is storage abstraction somewhat similar to a basic data file, with several key differences. The basic idea is that the application will create, read, and write to a tagline in fixed sized blocks. (See `tagline_driver.h`)
  - The "name" of a tagline is a 16-bit unsigned integer called the tag. This is assigned by the application and managed by your device driver.
  - The tagline consists of a zero-indexed set of blocks that are read from and written to. Taglines initially have zero blocks.
  - taglines are monotonically increasing, meaning that they are never deleted and they never shrink. Blocks can be overwritten however.
Tagline illustrated

WRITE(tag=1, blks=2, start=0)

WRITE(tag=1, blks=4, start=1)

READ(tag=1, blks=3, start=2)

WRITE(tag=1, blks=2, start=2)
Tagline calls (to your code)

• You are to implement the following functions:

  int tagline_driver_init(uint32_t maxlines);
  int tagline_read(TagLineNumber tag,
                  TagLineBlockNumber bnum, uint8_t blks,
                  char *buf);
  int tagline_write(TagLineNumber tag,
                    TagLineBlockNumber bnum, uint8_t blks,
                    char *buf);
  int tagline_close(void);

• Key to assignment: how do you convert each of these functions into the corresponding object calls and buffer manipulation?
RAID Array

- You are to build a software layer that implements the tagline interface on top of the raid array system.
- The raid array is a collection of disks each with a set of fixed sized blocks that are written and read. Notes:
  - Each block is of size RAID_BLOCK_SIZE
  - For this assignment, you can assume that there are RAID_DISKS disks, each with RAID_DISKBLOCKS blocks
  - The blocks are indexed 0 to RAID_DISKBLOCK-1

![RAID Array Firmware Diagram]
Using the RAID Array

- The array is driven by bus commands sent through:

```
RAIDOpCode raid_bus_request(RAIDOpCode request, void *buf);
```

- where the request and response are send in RAIDOpCode representing the data of the request:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>request type</td>
</tr>
<tr>
<td>8-15</td>
<td>number of blocks</td>
</tr>
<tr>
<td>16-23</td>
<td>disk number</td>
</tr>
<tr>
<td>24-30</td>
<td>unused (for now, always set to zero)</td>
</tr>
<tr>
<td>31</td>
<td>status this is the result bit (0 success)</td>
</tr>
<tr>
<td>32-63</td>
<td>block ID</td>
</tr>
</tbody>
</table>

![64-bit integer diagram]
RAID Commands

- RAID_INIT - Initialize the RAID interface
- RAID_FORMAT - Format a disk in the array. This will zero the disk contents.
- RAID_READ - Read consecutive blocks in the disk array
- RAID_WRITE - Write consecutive blocks to the disk array
- RAID_CLOSE – closes the array
What now?

• The real trick to all of this is that you have to:
  ‣ Initialize the disk array by calling the init & format functions
  ‣ On writes for new blocks, find a location on the disk to put it
    • Then call the RAID commands to get it to store the data
  ‣ On writes for old blocks, figure out where you put it
    • Then call the RAID commands to get it to overwrite the data
  ‣ On reads, return the previously stored data in those blocks
  ‣ Close the array when you are done

• Note, you are done when you see:

  [INFO] Tagline simulation completed successfully.
Simplifications

• Device driver will only receive requests for a single tagline (you only have to worry about one tagline)
  ‣ Any other tagline number other than the first you can return an error

• A tagline reads and writes will be for a single block
  ‣ You can return an error when any request asks for more or less than a single block
Getting started ...

- Get the file from the web:

  wget http://www.cse.psu.edu/~mcdaniel/cmpsc311-f15/docs/assign2-starter.tgz

- Change into your development directory and unpack the file:

  % cd ~/cmpsc311
  % cp assign2-starter.tgz cmpsc311
  % cd cmpsc311
  % tar xvfz assign2-starter.tgz
  % cd assign2
  % make

- Note: you may have to install libgcrypt-dev and libcurl4-gnutls-dev via apt-get.
Hints

• Use the logMessage interface to log information about how your program is running.
• Carefully read and understand the error messages that are being written to the log.
• Review the unit test function to see how the interfaces in the program should operate.
First functions …

- The first functions you should write should create a request structure and extract a request structure:

  ```c
  RAIDOpCode create_raid_request(...) {
      ???
  }

  ??? extract_raid_response(RAIDOpCode resp, ...) {
      ???
  }
  
  Hint: use the bit operations to build up structure ….