Assignment 2 - Back Store

Due: Monday March 14th, 11:59 AM (right before class)

Base tag: A2_BASE **Submission tag: A2**

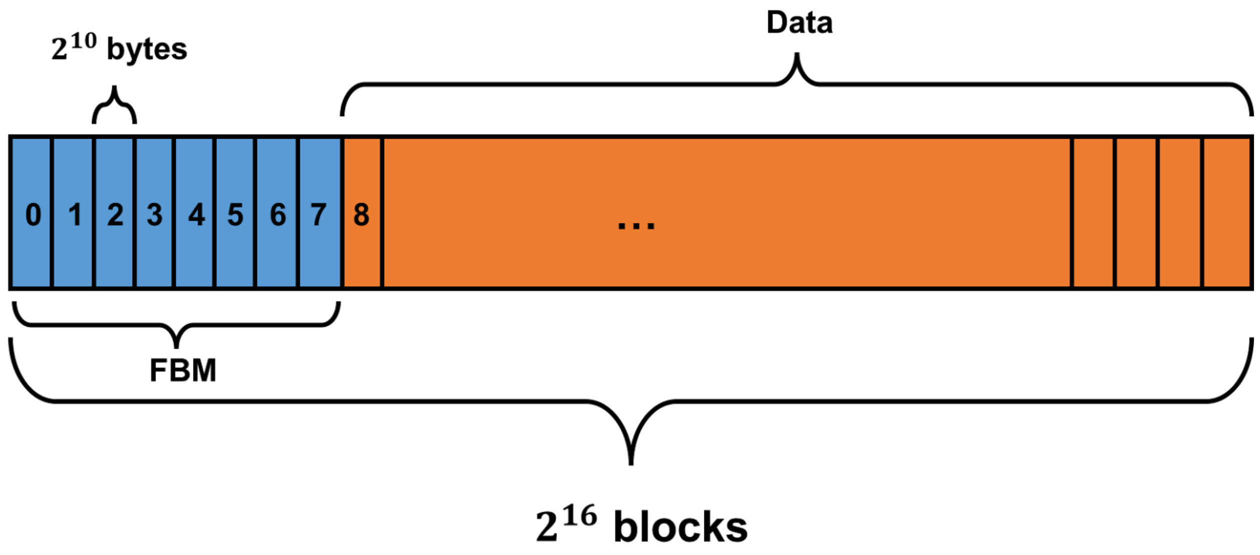
Objectives:

- **Become familiar with the following concepts:**
    - **Modular programming**
    - **Bitmaps, Bytes, and Blocks**
    - **Block storage devices**
    - **Free block maps**

Specifications:

You are to implement a storage library capable of storing blocks of data for the user. This data is to be file-backed, i.e. the stored data is written to disk so that it may be used later. Your implementation should create opaque, self-contained objects.

This exercise touches on many concepts you may be unfamiliar with, so be sure to read this document completely. Questions and requests for clarification should be directed at the course discussion board within the Canvas LMS.



Modular Programming:

1. Modular Programming and proper organization
2. Documentation of Code via thorough comments
3. Parameter testing and error checking

Bitmaps:

A bitmap is a data structure that can represent a large set of boolean data in a compact way. A boolean value can be represented with a single bit, but the smallest addressable unit is an entire byte, which leaves a lot of wasted space. If you need to represent multiple boolean values, you're going to waste even more memory, and this can add up quickly. This leaves less memory for other programs, which is a terrible thing for an OS or service to do.

A bitmap uses binary operations to store multiple boolean variables in a single byte, reducing your memory footprint to ⅛ of what it would be otherwise. In doing this, however, a bitmap has to use a more complex addressing system that uses both a bit and byte address in addition to masking.

Please Review: https://github.com/mizzoucs/OS_SP16_Learning_Modules/tree/master/bits
Resource Link: https://en.wikipedia.org/wiki/Bit_array

Block Storage Devices:

A block storage device is exactly what it sounds like: a device that stores blocks of data. Most storage devices (hard disk, optical disc, and flash drives) are block devices with varying block sizes (the number of bytes per block). Reading and writing data in blocks reduces overhead and allows for better data handling. Each block is referenced by an id number starting from 0 to N - 1 number of blocks, and these blocks are contiguous, essentially making a block device a giant physical array.

Please Review: https://github.com/mizzoucs/OS_SP16_Learning_Modules/tree/master/arrays
Resource Link: https://en.wikipedia.org/wiki/Block_(data_storage)

Block Maps:

A block map is how block storage devices keep track of important metadata. With this back store library, we will be using the Free Block Map (FBM), **implemented as a bitmap** to keep track of important device data.

The FBM keeps track of which blocks are currently in use. A bit value of one means the block is in use, and zero means the block is free. When a user requests a block of space, the library scans the FBM, finds a free block, marks that block as in use in the FBM, and returns the block id number for the user to use. When a block is freed, the corresponding bit in the FBM is cleared. Without this, devices would have no way of tracking which blocks are in use, and current data present is easily overwritten. The FBM is stored in the first few blocks of a device.

Resource Link: https://en.wikipedia.org/wiki/Block_allocation_map

Assignment Requirements:

1) You must update the given CMakeLists.txt to build a shared object (A.K.A. a dynamic library object) called libback_store.so that uses src/back_store.c and include/back_store.h for source files.
2) Complete all the blank functions inside src/back_store.c to build a library called **libback_store.so**

libback_store is a library that creates and manages an in-file block storage device. This version of libback_store has 8 public functions that need to be completed. The back store consists of 64K blocks of 1K bytes each, and therefore has an FBM size of 8K. The FBM makes up the first 8 blocks, which are considered reserved not allowed to be touched by the end user.

Rubric:

This assignment is out of 75 points, the tester is worth 65 (points are broken down per test) and 10 points for successful CMakeLists.txt update to build a library.

All file functions should be using POSIX interface, you **WILL** lose points for using FILE objects. If your code does not compile, you **WILL** receive a zero for the assignment.

Insufficient Parameter Validation -20% of rubric score
Insufficient Error Checking -20% of rubric score
Insufficient Block and Inline Comments -20% of rubric score
Submission compiles with warnings (with -Wall -Wextra -Wshadow) -80%
Submission does not compile -100% of rubric score