Virginia Tech ❖ Bradley Department of Electrical and Computer Engineering

ECE 4984 / 5984 Linux Kernel Programming
Spring 2017

On-flash, in-kernel key-value store

1 Introduction

In this project you are going to develop a non-volatile storage system in the kernel. This storage system is a simple key-value store. You will be given the sources of a prototype for this system. That prototype has some serious limitation that you will be asked to identify in the first phase of the project, by studying the code of the prototype. In the next phase, you will enhance the prototype by designing and implementing solutions to these limitations. You will then validate your implementation and evaluate its performance.

The key-value store has the particularity of using flash memory as the storage medium. Flash memory exhibits some specific constraints in its operation, and these constraints will need to be taken into account in the development of your solution. In this project we will use the *nandsim* flash memory simulator to emulate a flash memory chip.

This project involves the following topics:

- Device drivers;
- Storage management;
- Interface with user-space;
- Flash memory.

This is a group project and it should be done by groups of up to 4 students. Please use canvas to register your group.

2 Project description

2.1 Project steps

1. The first step that should be accomplished in the first days of the project is to **study the prototype and identify its limitations**. Studying the textual description of the prototype, presented in the annex of this document, should be sufficient to identify limitations. Studying the code is also important to understand the environment in which your future implementation will take place.

   In order to identify the limitations, you need to ask yourself the following questions:

   - *What are the usual functionalities of a storage system that are absent in the prototype?*
   - *What is hindering the performance and the scalability of the prototype*?

   Each group should send a (small) email to the course instructor containing a list of identified limitations and for each of these a one/two sentences description. This email should be sent by **04/06 10AM**, and the limitation will be reviewed by the entire class during a lecture session on **04/06**. To fully understand how to operate flash memory, a full lecture session will be dedicated to the subject on 04/04.

   As a reminder, the instructor email is: `polivier@vt.edu`

2. You will then be asked to modify the prototype in order to solve a list of limitation:

- One critical limitation will be mandatory to solve by every group (it will be disclosed on 04/06);
- Two additional limitation should be solved, picked up from a list of secondary limitations also disclosed on 04/096.

3. The last step of the project is the validation and evaluation of the performance and the scalability of the implementation.

2.2 Project report

A report of 10 pages minimum should be produced as one of the results to be handed by the project deadline. It should contain:

1. A description of the limitations solved;
2. A presentation of the design and the implementation of the solution, with justifications for important design choices;
3. A description of the validation process as well as the evaluation section describing the implementation validation and its performance evaluation process:
   - Performance evaluation scenarios description;
   - Measured results presentation and interpretation.
4. Optionally, a textual description of future works:
   - Potential additional functionalities design ideas;
   - Potential ideas on how to further enhance the performance.

3 Results to be handed - Deadline: 05/10 11:59PM

The following is expected to be handed by 05/10:

1. Project report in PDF format;

2. Sources of all developed software: storage system, potential validation/benchmarking scripts, etc., with brief documentation about the corresponding software usage.

All of this should be contained in an archive. One archive per group should be submitted.

Annex

A Flash memory basics

The prototype is directly manipulating the flash memory through the corresponding Linux driver. Here, some basic information about flash memory is given first, before describing the prototype design. Note that there will be a full lecture session about flash memory on 04/04.

Structure. As depicted in Figure 1, a flash memory chip contains a given number of *blocks*, each block itself containing *pages*. As an example, a 256MB flash memory chip can contain 2048 blocks with 64 2KB-sized pages each. Number and sizes of blocks/pages vary with the flash chip model.

Operations. 3 main operations are supported at the chip level: *read* and *write* operations are performed on a whole flash page. The *erase* operation targets an entire block. In terms of latencies, examples are 25μs for read, 200μs for write, and 500μs for erase.

Constraints: Some specific constraints need to be respected when operating the flash memory chip:

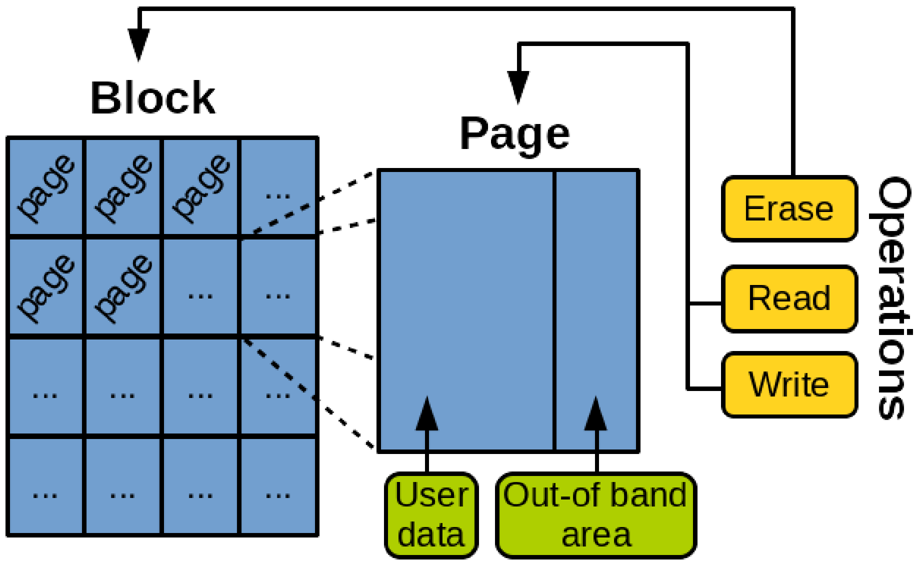

Figure 1: A flash memory chip internal organization

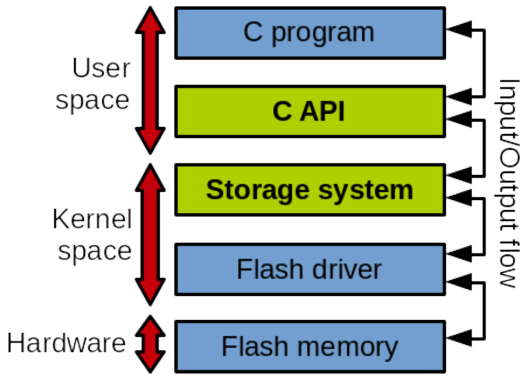

Figure 2: Key-value store location in the kernel hardware/software stack.

1. The write operation can only be performed in a previously erased flash page. If one wants to write in a non-erased page (i.e. a flash page containing data), it is first needed to erase this page, which means erasing the entirety of the containing block;

2. A given block can only sustain a limited number of erase operations. After a threshold is reached (between 10^4 and 10^5 according to the chip model), the block is worn-out and cannot be used anymore;

3. Pages in a block must be written sequentially (from the first to the last one). They can be read in any order.

It is indispensable to take these constraints into account in the development of your solution.

B Prototype description

General presentation. The integration of the prototype in the Linux storage software and hardware stack is presented in Figure 2. The core of the storage system is implemented as a kernel module and is using the flash driver to access the flash memory chip. The driver provides an API to perform flash page read and write, as well as block erase operations.

The storage system exports a virtual device in the `/dev` directory, which is used by user-space application to request services from the key-value store. Operations on that file are wrapped by a C library that offer simple

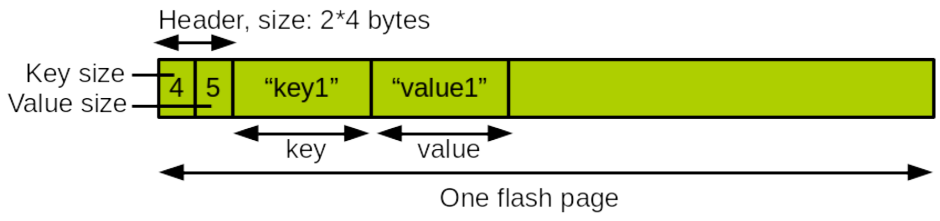

Figure 3: Key-value storage format inside a flash page

interfaces to access the key-value store. This library is made to be included in regular user programs wanting to access the key-value store.

Storage format. The prototype only manages keys and value that are character strings. Each flash page stores one and only one key-value couple. Storage is realized in each page as depicted on Figure 3.

A *header* containing two integer values, each of them coded on 4 bytes, is stored in the first eight bytes of the flash page. The first integer contains the size of the key in bytes (i.e. in characters). The second one contains the size of the associated value. The key is stored after the header, followed by the value.

The prototype supports three main operations: inserting a new key-value couple (*set*), retrieving a value from a key (*get*) and erasing the entire flash memory (*format*).

Writing a key-value couple. The prototype implements a *set* function taking two character arrays as parameters, representing the key/value couple to store. It is written in a free flash page according to the previously presented format. Selecting the flash page to write is made as follows: a free block (i.e. an erased block) is selected an its pages are written sequentially as new keys and values are stored. Once the block is full, a new free block is chosen. When there is no free block left, the storage system switches to read-only mode: it is not possible to perform *set* operations anymore.

Reading a value from a key. The prototype implements a *get* function taking as a parameter a character array, a key, and returning the corresponding value if the key is found on the storage system. The prototype implementation scans the entirety of the non-free blocks of the flash memory searching for the key. If the key is found, the corresponding value is returned.

Erasing the entire flash memory. The prototype implements a function that erase the entirety of the blocks of the managed partition: all stored keys and values are lost.

C Using the storage system from user-space

The set, get and format function are implemented by the module in kernel space, and thus they are not directly accessible from user-space. In order to establish the connection with user-space, the prototype uses a mechanism named `ioctl` provided by Linux in the form of a system call (`ioctl()`) applied on a virtual device file present in `dev`. That file the interface to communicate with the storage system from user-space.

This is implemented as follows: when the prototype module is inserted, a virtual device is created inside the operating system, represented by a file: `/dev/lkp_kv`. User-space programs can then call the `ioctl()` system call on that file to send commands (get, set, format) with associated parameters (keys and/or values) get the results from these commands (values, return codes).

Using `ioctl` from user-space to send such a command can be somehow tedious: it implies opening the virtual device file, preparing the data structure corresponding to the command parameters, and processing the data structure corresponding to the results, amongst other things. To simplify the storage system usage from user-space, ioctl processing is encapsulated inside a C library (a C file and the corresponding header file) intended to be linked with a regular user-space program wanting to access the storage system. The corresponding API, exported in the header file, present simple interfaces for each operation (get, set, format).

D Prototype source code

The source code is availabe here: `http://bit.ly/2nnWn6E`. This archive contains the user and kernel level code of the prototype. In addition, some scripts are provided in order to launch the *nandsim* flash simulator. A simple test application is also present in the archive.

A virtual machine containing the *nandsim* flash simulator installed as a module is available here: `http://bit.ly/2oqs3dl`.