# 实验六、综合设计实验

## 实验目的

基于BlobFS，设计一个key-value数据库，支持open，put，get，close操作

## 实验内容

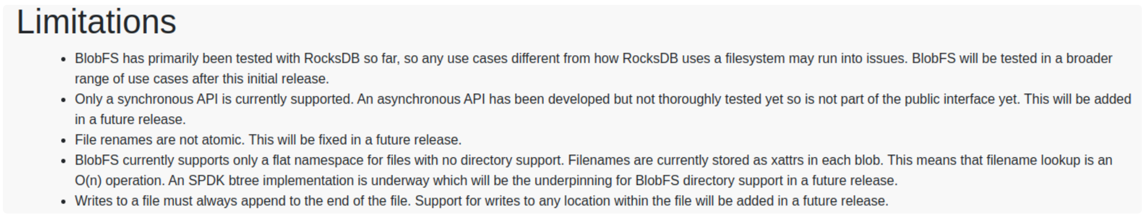
基于BlobFS，设计一个key-value数据库，支持open，put，get，close操作

## 实验过程和步骤

### 实验思路

本次基于BlobFS设计KV数据库。考虑到BlobFS仅支持追加写的特性，本次实验选择使用C++复现Bitcask数据库。

- BlobFS的限制



Limitations

- BlobFS has primarily been tested with RocksDB so far, so any use cases different from how RocksDB uses a filesystem may run into issues. BlobFS will be tested in a broader range of use cases after this initial release.
- Only a synchronous API is currently supported. An asynchronous API has been developed but not thoroughly tested yet so is not part of the public interface yet. This will be added in a future release.
- File renames are not atomic. This will be fixed in a future release.
- BlobFS currently supports only a flat namespace for files with no directory support. Filenames are currently stored as xattrs in each blob. This means that filename lookup is an O(n) operation. An SPDK btree implementation is underway which will be the underpinning for BlobFS directory support in a future release.
- Writes to a file must always append to the end of the file. Support for writes to any location within the file will be added in a future release.

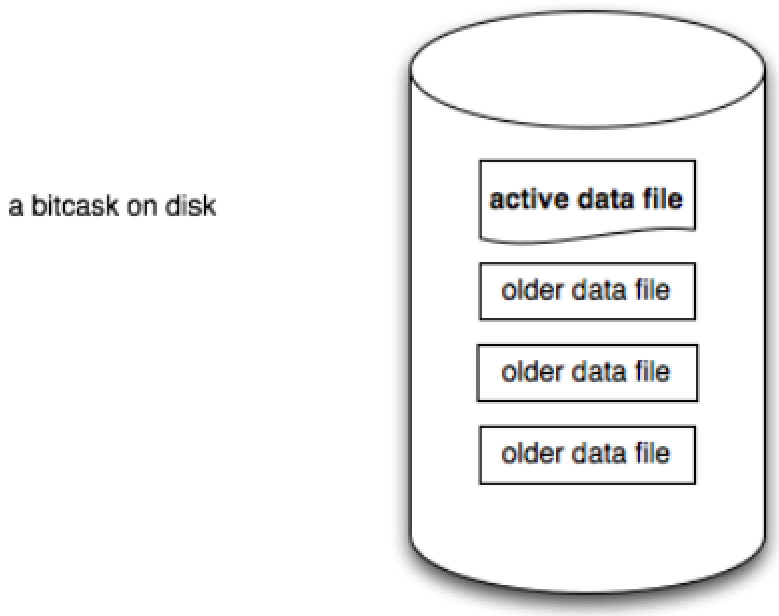
### Bitcask数据库原理

#### 日志型数据存储

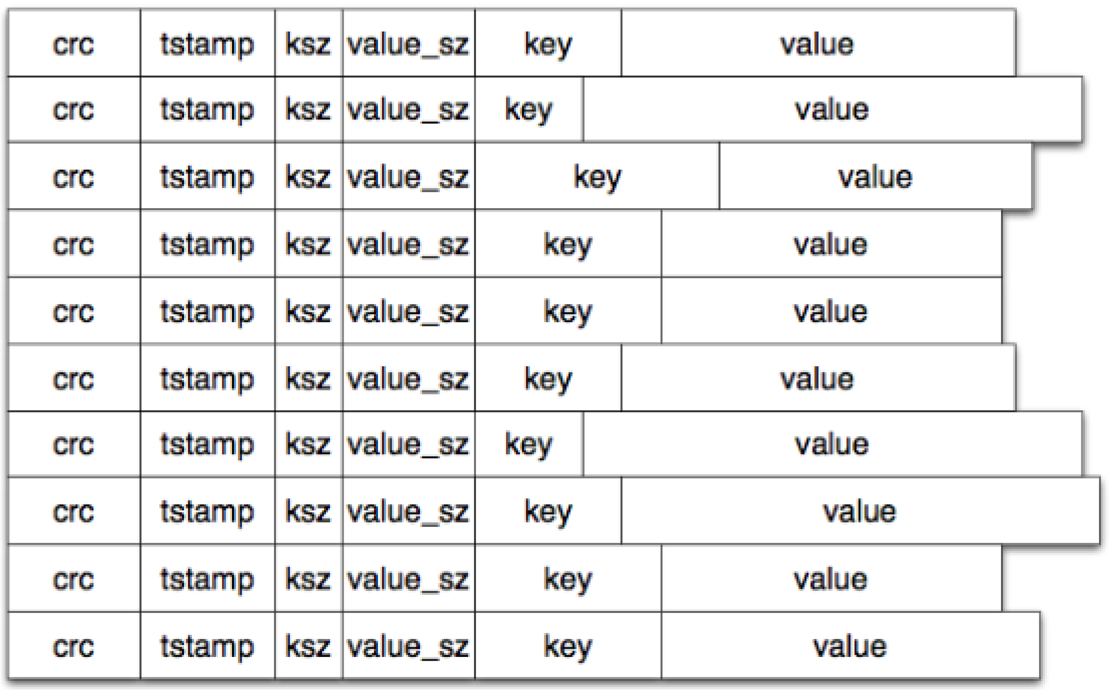
所有写操作只追加而不修改老的数据，这样做目的是能保证最大程度的顺序 IO ，压榨出机械硬盘的顺序写性能。

在Bitcask模型中，数据文件以日志型只增不减的写入文件，而文件有一定的大小限制，当文件大小增加到相应的限制时，就会产生一个新的文件，老的文件将只读不写。

在任意时间点，只有一个文件是可写的，在Bitcask模型中称其为active data file，而其他的已经达到限制大小的文件，称为older data file，如下图：



文件中的数据结构非常简单，是一条一条的数据写入操作，每一条数据的结构如下：

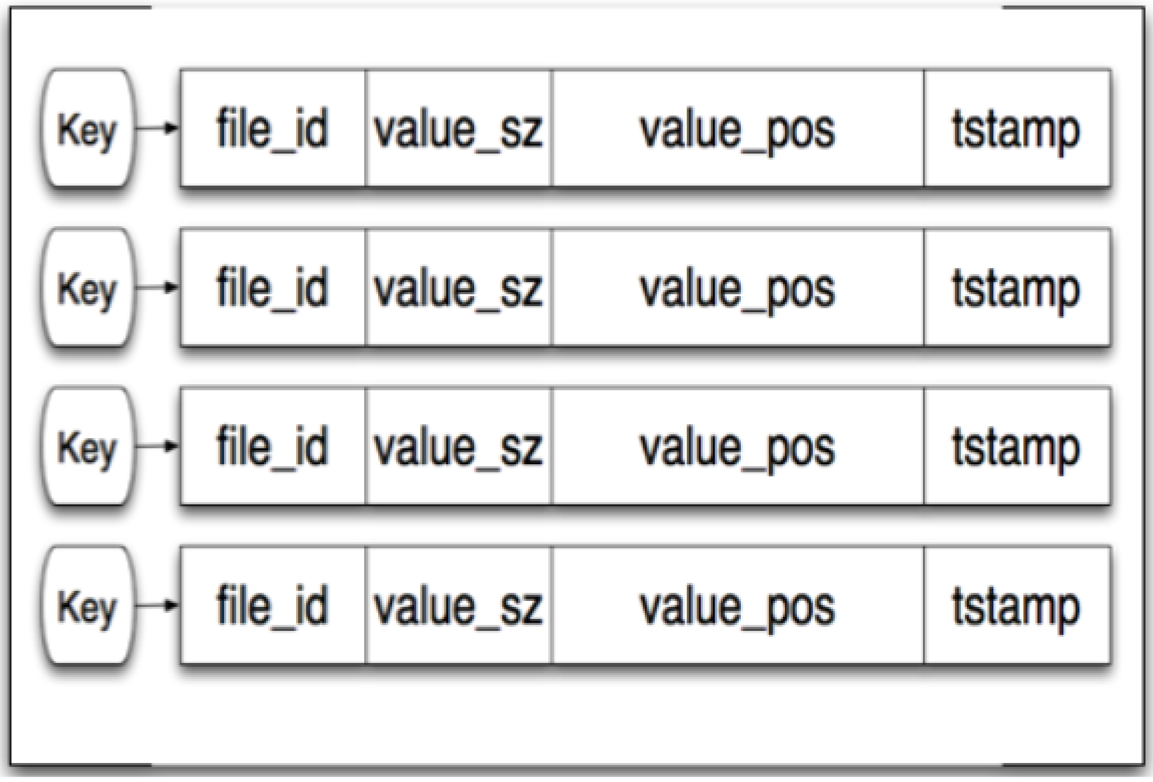


| | | | | | |
|---|---|---|---|---|---|
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |
| crc | tstamp | ksz | value_sz | key | value |

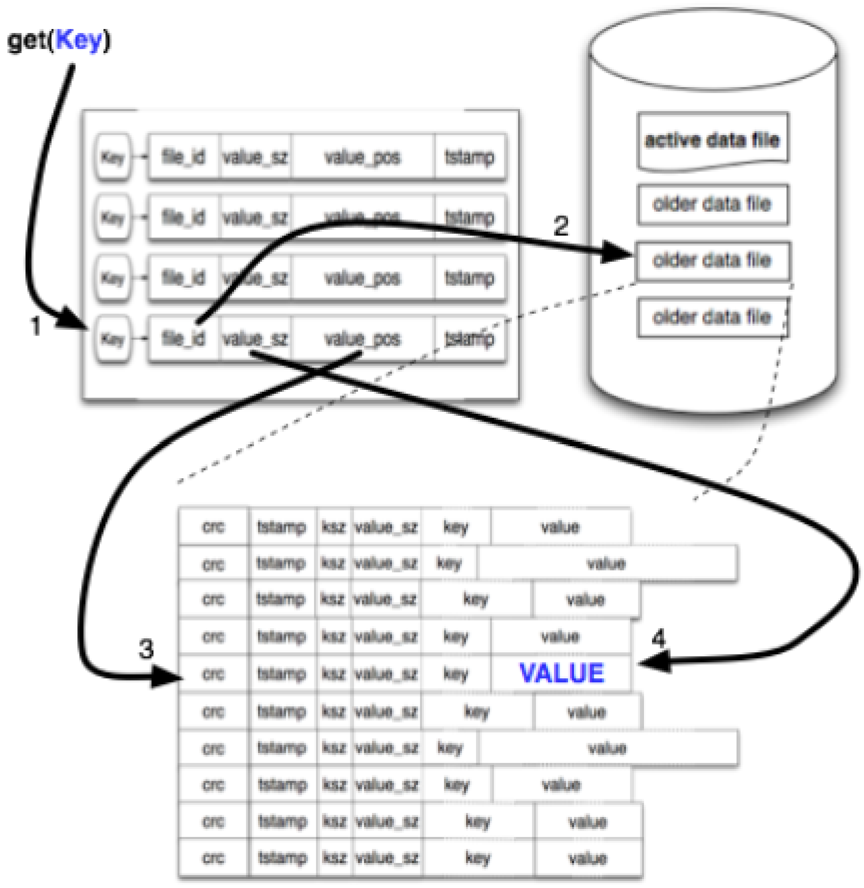
## 基于hash表的索引数据

日志类型的数据文件会让我们的写入操作非常快，而如果在这样的日志型数据上进行key值查找，那将是一件非常低效的事情。于是我们需要使用一些方法来提高查找效率。

例如在Bigtable中，使用bloom-filter算法为每一个数据文件维护一个bloom-filter 的数据块，以此来判定一个值是否在某一个数据文件中。
在Bitcask模型中，除了存储在磁盘上的数据文件，还有另外一块数据，那就是存储在内存中的hash表，hash表的作用是通过key值快速的定位到value的位置。hash表的结构大致如下图所示：



hash表对应的这个结构中包括了三个用于定位数据value的信息，分别是文件id号(file_id)，value值在文件中的位置（value_pos），value值的大小（value_sz），于是我们通过读取file_id对应文件的value_pos开始的value_sz个字节，就得到了我们需要的value值。整个过程如下图所示：



由于多了一个hash表的存在，我们的写操作就需要多更新一块内容，即这个hash表的对应关系。于是一个写操作就需要进行一次顺序的磁盘写入和一次内存操作。

## 代码实现

`bitcask.h`

```
#ifndef BITCASK_H_
#define BITCASK_H_

#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <pwd.h>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/archive/binary_iarchive.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/list.hpp>
#include <boost/crc.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/date_time/posix_time/time_serialize.hpp>

using namespace std;

#define filemax 1024 * 4096 // 4MB

const string fileprev = "bitcask_data";
const string cmd_prompt = " >>> bitcask : ";
const string cmd = " >>> ";
const int number = 0;
// data
struct bitcask_data
{
    string key;
    int key_len;
    string value;
    int value_len;
    boost::posix_time::ptime timestamp;
    uint32_t crc;
    // crc
    template <typename Archive>
    void serialize(Archive &ar, const unsigned int version)
    {
        ar &key_len;
        ar &key;
        ar &value_len;
        ar &value;
        ar &crc;
        ar &timestamp;
    }
};

// index
struct bitcask_index
```

```
{
    string key;
    string file_id;
    int value_pos;
    int value_len;
    boost::posix_time::ptime timestamp;
    bool value_valid;

    template <typename Archive>
    void serialize(Archive &ar, const unsigned int version)
    {
        ar &key;
        ar &file_id;
        ar &value_pos;
        ar &value_len;
        ar &timestamp;
        ar &value_valid;
    }

    bitcask_index()
    {
        value_valid = false;
    }
};

// bitcask
class bitcask
{
private:
    unordered_map<string, bitcask_index> index;
    int _activefile;
    bool _start;
    bool _finish;
    string _response;
    string filepath;

private:
    void init(string path);
    uint32_t crc32(string value);
    void insert_data(string key, string value);
    void write_data(bitcask_data newdata);
    void write_index(bitcask_index newindex);
    bitcask_data read_data(string key);
    void read_datainfo(string key);
    bitcask_index read_index(string key);
    void delete_data(string key);
    void update_data(string key, string value);
    void update_index(bitcask_index upindex, string key);
    void merge();
    void flush(); // flush index :hint.bin

public:
    bitcask();
    string Get(string key);
    void Put(string key, string value);
    void Open(string path);
    void Close();
    ~bitcask();

};

#endif /* BITCASK_H_ */
```

---

`bitcask.cpp`

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
#include <unistd.h>
#include <dirent.h>
#include <pwd.h>
#include "bitcask.h"
using namespace std;

bitcask::bitcask()
{
    _start = false;
    _activefile = 0;
    _finish = false;
    _response = "";
    filepath = "";
}

bitcask::~bitcask()
{
    if (this->_start)
    {
        merge();
        flush();
    }
}

void bitcask::init(string path)
{
    this->filepath = path;
    this->_start = true;
    long len;
    fstream hint;
    hint.open(filepath + "hint.bin", ios::binary | ios::out | ios::app);
    if (!hint)
    {
        cout << "the file hint.bin open failure or maybe not exist!\n";
    }
    len = hint.tellg();
    if (len == 0)
    {
        cout << "create file hint.bin successful!\n";
    }
    else
    {
        /*
         load index to memory
        */
        // cout << "loading index" << endl;
        bitcask_index search;
        fstream hint;
        hint.open(filepath + "hint.bin", ios::binary | ios::in);
        if (!hint)
        {
            cout << "the file hint.bin open failure or maybe not exist!\n";
        }

        while (hint)
        {
            boost::archive::binary_iarchive ia(hint,
boost::archive::no_header);
            try
            {
                ia >> search;
            }
            catch (const exception &e)
            {
                // cout << "read end" << endl;
                goto do_load;
            }
            // cout << "loading:" << search.key << endl;
            bitcask_index insert;
            insert.file_id = search.file_id;
            insert.value_pos = search.value_pos;
            insert.value_len = search.value_len;
            insert.timestamp = search.timestamp;
            insert.value_valid = search.value_valid;
            index[search.key] = insert;
        }
    }
do_load:
    fstream filelog;
    filelog.open(filepath + "filelog.bin", ios::binary | ios::in);
    if (!filelog)
    {
        cout << "the file filelog.bin open failure or maybe not exist!\n";
    }
    filelog.read((char *)(&_activefile), sizeof(int));
    filelog.close();
    if (_activefile == 0)
    {
        cout << "create file filelog.bin successful!\n";
        _activefile = 1;
        filelog.open(filepath + "filelog.bin", ios::binary | ios::out |
ios::app);
        filelog.write((char *)(&_activefile), sizeof(int));
        filelog.close();
        return;
    }
    _start = true;
}

uint32_t bitcask::crc32(string value)
{
    boost::crc_32_type result;
    result.process_bytes(value.c_str(), value.length());
    return result.checksum();
}

void bitcask::insert_data(string key, string value)
{
    bitcask_index search = read_index(key);
    if (search.key != "")
    {
        cout << "the data " + key + " already exist!\n";
        return update_data(key, value);
    }
    // add data
    bitcask_data newdata;
    newdata.key = key;
    newdata.key_len = int(key.length());
    newdata.value = value;
    newdata.value_len = int(value.length());
    newdata.crc = crc32(value);
    // newdata.timestamp=time(0);
    newdata.timestamp = boost::posix_time::microsec_clock::universal_time();

    // add index
    fstream datafile;
    bitcask_index newindex;
    newindex.key = key;
    newindex.file_id = fileprev + to_string(_activefile);
    datafile.open(filepath + newindex.file_id, ios::binary | ios::out |
ios::app);
    if (!datafile)
        cout << cmd_prompt + newindex.file_id + " open failure\n";
    newindex.value_pos = datafile.tellg();
    if (newindex.value_pos > filemax || filemax - newindex.value_pos <
sizeof(newdata))
    {
        _activefile++;
        newindex.file_id = fileprev + to_string(_activefile);
    }
    newindex.timestamp = newdata.timestamp;
    newindex.value_len = sizeof(newdata);
    newindex.value_valid = true;
    datafile.close();
    write_data(newdata);
    write_index(newindex);
    // add to memory index array
    index[key] = newindex;
    // cout << "the data " + key + " insert successful\n";
}

void bitcask::write_data(bitcask_data newdata)
{
    string file = fileprev + to_string(_activefile);
    fstream datafile;
    datafile.open(filepath + file, ios::binary | ios::out | ios::app);
    if (!datafile)
        cout << file + " open file " + file + " failure!\n";
    // datafile.write((char *)(&newdata),sizeof(newdata));
    // data_append(datafile, newdata);
    boost::archive::binary_oarchive oa(datafile, boost::archive::no_header);
    oa << newdata;
    datafile.close();
}

void bitcask::write_index(bitcask_index newindex)
{
    fstream hint;
    hint.open(filepath + "hint.bin", ios::binary | ios::out | ios::app);
    if (!hint)
    {
        cout << "the file hint.bin open failure!\n";
    }
    // hint.write((char *)(&newindex), sizeof(newindex));
    // cout << "writing index: " << newindex.key << newindex.timestamp <<
endl;
    boost::archive::binary_oarchive oa(hint, boost::archive::no_header);
    oa << newindex;
    hint.close();
}

bitcask_index bitcask::read_index(string key)
{

    bitcask_index search;
    if ("" == key)
    {
        return search;
    }
    for (auto indexinfo : index)
    {
        if (indexinfo.first == key)
        {
            {
                search.key = indexinfo.first;
                search.file_id = indexinfo.second.file_id;
                search.value_pos = indexinfo.second.value_pos;
                search.value_len = indexinfo.second.value_len;
                search.value_valid = indexinfo.second.value_valid;
                search.timestamp = indexinfo.second.timestamp;
                // cout << "got key for " << key << " ,valid " <<
search.value_valid << endl;
                return search;
            }
        }
    }
    return search;
}

bitcask_data bitcask::read_data(string key)
{
    bitcask_data search_data;
    bitcask_index search_index = read_index(key);
    if (search_index.value_valid == true)
    {
        string file = search_index.file_id;
        // cout << "reading file" << file << endl;
        fstream datafile;
        datafile.open(filepath + file, ios::binary | ios::in);
        if (!datafile)
            cout << "open file " + file + " failure\n";

        boost::archive::binary_iarchive ia(datafile,
boost::archive::no_header);
        if (search_index.value_pos)
        {
            datafile.seekg(search_index.value_pos, ios::beg);
            // cout << "seeking to" << search_index.value_pos << endl;
        }
        // datafile.read((char *)(&search_data),sizeof(search_data));
        // cout << "reading datafile" << filepath + file << endl;
        // cout << "key = " << search_index.key << endl;
        // cout << "value_pos = " << search_index.value_pos << endl;
        // cout << "ts = " << search_index.timestamp << endl;
        // cout << "file pos = " << datafile.tellp() << endl;
        // search_data.serialize(ia, 0);
        ia >> search_data;
        // cout << "file pos = " << datafile.tellp() << endl;
        // cout << "on-disk ts = " << search_data.timestamp << endl;
        // return search_data;
    }
    // else
    //     cout << "the data " + key + " does not exist!\n";
    return search_data;
}

void bitcask::read_datainfo(string key)
{
    bitcask_data data = read_data(key);
    bitcask_index index = read_index(key);
    if (index.value_valid == true)
    {
        _response += cmd + "key :" + key + "\n";
        _response += cmd + "value :" + data.value + "\n";
        _response += cmd + "crc :" + to_string(data.crc) + "\n";
        _response += cmd + "file id :" + index.file_id + "\n";
        _response += cmd + "value pos :" + to_string(index.value_pos) + "\n";
        _response += cmd + "value length :" + to_string(data.value_len) +
"\n";
        // cout << _response;
        //_response+=cmd+"time :"+ data.timestamp;
        //_response+=cmd+"time :"+ctime(&data.timestamp);
    }
    else
        return;
}

void bitcask::delete_data(string key)
{
    bitcask_index delindex = read_index(key);
    if (delindex.key != "")
    {
        delindex.value_valid = false;
        index[key] = delindex;
        cout << "the data " + key + " already delete!\n";
    }
    else
        _response += cmd_prompt + "the data " + key + " does not exist!\n";
}

void bitcask::update_data(string key, string value)
{

    bitcask_index search = read_index(key);
    if (search.key == "")
    {
        cout << "the data " + key + " does not exist!\n";
        return;
    }
    // update data
    fstream datafile, hintfile;
    bitcask_data updata;
    bitcask_index upindex = read_index(key);
    updata.key = key;
    updata.key_len = int(key.length());
    updata.value = value;
    updata.value_len = int(value.length());
    // updata.timestamp=time(0);
    updata.timestamp = boost::posix_time::microsec_clock::universal_time();

    // update index
    upindex.file_id = fileprev + to_string(_activefile);
    datafile.open(filepath + upindex.file_id, ios::binary | ios::in |
ios::app);
    if (!datafile)
        _response += cmd_prompt + "the file " + upindex.file_id + " open
failure!\n";
    upindex.value_pos = datafile.tellg();
    if (upindex.value_pos > filemax || filemax - upindex.value_pos <
sizeof(updata))
    {
        _activefile++;
    }
    upindex.timestamp = updata.timestamp;
    upindex.value_len = sizeof(updata);
    upindex.value_valid = true;
    datafile.close();
    write_data(updata);
    update_index(upindex, key);
    _response += cmd_prompt + "the data " + key + " update successful\n";
}

void bitcask::update_index(bitcask_index upindex, string key)
{
    bitcask_index seaindex = read_index(key);
    seaindex.key = key;
    seaindex.file_id = upindex.file_id;
    seaindex.value_pos = upindex.value_pos;
    seaindex.value_len = upindex.value_len;
    seaindex.value_valid = upindex.value_valid;
    seaindex.timestamp = upindex.timestamp;
    index[key] = seaindex;
}

void bitcask::merge()
{
    /*
     merge data in file
     function: delete data in file
     */
    int beans = 1;
    long value_pos;
    vector<bitcask_data> data_array;
    // cout << "merge begin: activefile" << _activefile << endl;
    for (; beans <= _activefile; beans++)
    {

        string file = fileprev + to_string(beans);
        fstream datafile;
        datafile.open(filepath + file, ios::binary | ios::in);

        if (!datafile)
        {
            _response += cmd_prompt + "the data file " + file + " open
failure!\n";
        }
        bitcask_data beans_data;
        bitcask_index beans_index;
        datafile.seekg(0, ios::beg);
        // load to memory
        while (datafile)
        {
            boost::archive::binary_iarchive ia(datafile,
boost::archive::no_header);
            try
            {
                ia >> beans_data;

                beans_index = read_index(beans_data.key);
                // cout << "ts:" << beans_data.timestamp << " vs " <<
beans_index.timestamp << endl;
            }
            catch (const exception &e)
            {
                goto do_merge;
            }

            if (beans_index.value_valid == true && beans_data.timestamp ==
beans_index.timestamp)
            {
                // cout << "pushing:" << beans_index.key << endl;
                data_array.push_back(beans_data);
            }
        }

    do_merge:
        for (auto data : data_array)
        {
            // cout << "dataarray = " << data.key << endl;
        }

        datafile.close();
        // write to file
        datafile.open(filepath + file, ios::binary | ios::out);
        if (!datafile)
        {
            _response += cmd_prompt + "the data file " + file + " open
failure!\n";
        }
        datafile.seekg(0, ios::beg);

        for (auto data : data_array)
        {
            value_pos = datafile.tellg();
            // datafile.write((char *)(&data),sizeof(data));
            boost::archive::binary_oarchive oa(datafile,
boost::archive::no_header);
            oa << data;
            bitcask_index seaindex = read_index(data.key);
            seaindex.value_pos = value_pos;
            index[data.key] = seaindex;
        }
        datafile.close();
        data_array.clear();
    }
    /*
     TODO merge file
     */
    if (_activefile >= 2)
    {
        while (_activefile > 1)
        {
            string file = fileprev + to_string(_activefile);
            fstream datafile;
            datafile.open(filepath + file, ios::binary | ios::in);
            if (!datafile)
            {
                _response += cmd_prompt + "the data file " + file + " open
failure!\n";
            }
            bitcask_data beans_data;
            bitcask_index beans_index;
            datafile.seekg(0, ios::beg);
            // TODO load to memory maybe too big?
            while (datafile)
            {
                boost::archive::binary_iarchive ia(datafile,
boost::archive::no_header);
                try
                {
                    ia >> beans_data;
                }
                catch (const exception &e)
                {
                    goto do_merge_2;
                }

                data_array.push_back(beans_data);
            }

        do_merge_2:
            datafile.close();
            // write to file
            for (int pos = 1; pos < _activefile; pos++)
            {
                string mergefile = fileprev + to_string(pos);
                fstream datafile;
                datafile.open(filepath + mergefile, ios::binary | ios::out |
ios::app);
                if (!datafile)
                {
                    _response += cmd_prompt + "the data file " + file + "
open failure!\n";
                }
                long mergefile_end = datafile.tellg();
                bitcask_data merge_data = data_array.back();
                bitcask_index merge_index = read_index(merge_data.key);
                while (mergefile_end < filemax && (filemax - mergefile_end) <
sizeof(merge_data))
                {
                    // datafile.write((char *)(&merge_data),
sizeof(merge_data));
                    boost::archive::binary_oarchive oa(datafile,
boost::archive::no_header);
                    oa << merge_data;
                    merge_data = data_array.back();
                    merge_index = read_index(merge_data.key);
                    merge_index.value_pos = mergefile_end;
                    merge_index.file_id = mergefile;
                    index[merge_data.key] = merge_index;
                    mergefile_end += sizeof(merge_data);
                    data_array.pop_back();
                    merge_data = data_array.back();
                }
                datafile.close();
            }
            if (data_array.size() == 0)
            {
                _activefile--;
            }
            else
            {
                fstream newdatafile;
                newdatafile.open(filepath + file, ios::binary | ios::out);
                if (!newdatafile)
                {
                    _response += cmd_prompt + "the data file " + file + "
open failure!\n";
                }
                newdatafile.seekg(0, ios::beg);
                for (auto data : data_array)
                {
                    // newdatafile.write((char *)(&data), sizeof(data));
                    boost::archive::binary_oarchive oa(newdatafile,
boost::archive::no_header);
                    oa << data;
                }
                newdatafile.close();
                break;
            }
        }
    }
}

void bitcask::flush()
{
    // write index file to index file hint.bin
    fstream hint;
    hint.open(filepath + "hint.bin", ios::binary | ios::out);
    if (!hint)
    {
        _response += cmd_prompt + "the file hint.bin open failure!\n";
    }
    hint.seekg(0, ios::beg);
    for (auto indexinfo : index)
    {
        if (indexinfo.second.value_valid == true)
        {
            // hint.write((char *)(&indexinfo.second),
sizeof(indexinfo.second));
            boost::archive::binary_oarchive oa(hint,
boost::archive::no_header);
            oa << indexinfo.second;
        }
    }
    // hint.close();
    // write active file number to file filelog.bin
    fstream filelog;
    filelog.open(filepath + "filelog.bin", ios::binary | ios::out);
    if (!filelog)
    {
        _response += cmd_prompt + "the filelog.bin open failuer!\n";
    }
    filelog.write((char *)(&_activefile), sizeof(int));
}

string bitcask::Get(string key)
{
    if (this->_start == true)
    {
        bitcask_data bc_data = read_data(key);
        if (bc_data.key.length() == 0)
            cout << key + " does not exist!" << endl;
        return bc_data.value;
    }
    else
    {
        cout << "please open a database first" << endl;
        return "";
    }
}

void bitcask::Put(string key, string value)
{
    if (this->_start == true)
        insert_data(key, value);
    else
        cout << "please open a database first" << endl;
}

void bitcask::Open(string path)
{
    if (this->_start == false)
        init(path);
    else
        cout << "already open a database, please close first" << endl;
}

void bitcask::Close()
{
    if (this->_start)
    {
        // merge();
        flush();
        _start = false;
        _activefile = 0;
        _finish = false;
        _response = "";
        filepath = "";
    }
    else
        cout << "please open a database first" << endl;
}
```

---

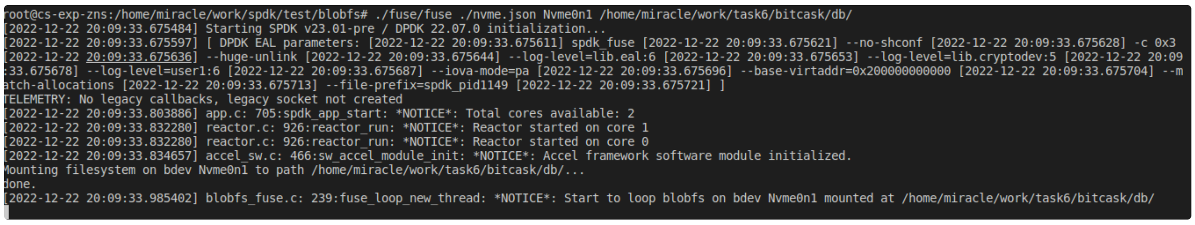
`main.cpp`

```
#include "bitcask.h"

#define db_path "/home/miracle/work/task6/bitcask/db/"

int main()
{
    bitcask *db = new bitcask;
    db->Open(db_path);

    db->Put("key1", "value1");
    cout << db->Get("key1") << endl;
    db->Close();
    cout << db->Get("key1") << endl;
    db->Open(db_path);
    cout << db->Get("key1") << endl;
    cout << db->Get("key2") << endl;
    return 0;
}
```

## 实验结果

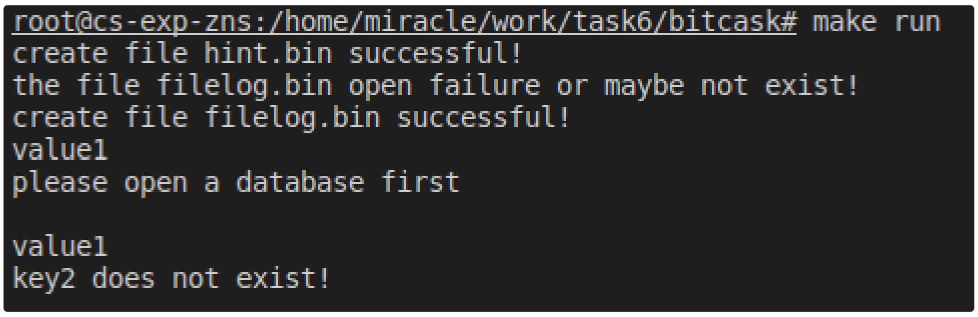
### 挂载BlobFS

将BlobFS挂载到当前目录下的db文件夹，用作数据库存储

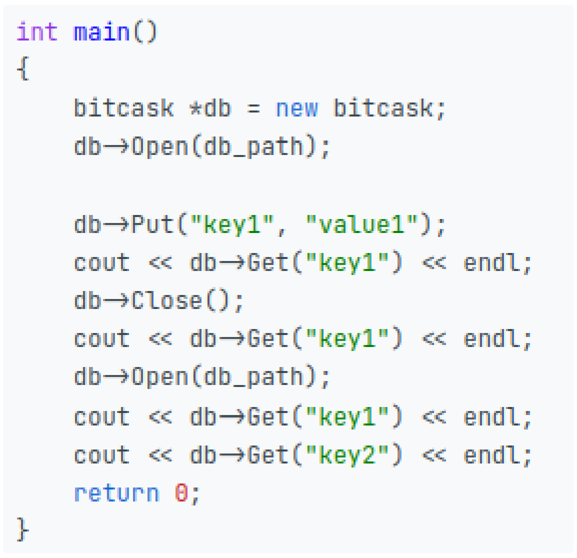


```
root@cs-exp-zns:/home/miracle/work/spdk/test/blobfs# ./fuse/fuse ./nvme.json Nvme0n1 /home/miracle/work/task6/bitcask/db/
[2022-12-22 20:09:33.675484] Starting SPDK v23.01-pre / DPDK 22.07.0 initialization...
[2022-12-22 20:09:33.675597] [ DPDK EAL parameters: [2022-12-22 20:09:33.675611] spdk_fuse [2022-12-22 20:09:33.675621] --no-shconf [2022-12-22 20:09:33.675628] -c 0x3
[2022-12-22 20:09:33.675636] --huge-unlink [2022-12-22 20:09:33.675644] --log-level=lib.eal:6 [2022-12-22 20:09:33.675653] --log-level=lib.cryptodev:5 [2022-12-22 20:09
:33.675678] --log-level=user1:6 [2022-12-22 20:09:33.675687] --iova-mode=pa [2022-12-22 20:09:33.675696] --base-virtaddr=0x200000000000 [2022-12-22 20:09:33.675704] --m
atch-allocations [2022-12-22 20:09:33.675713] --file-prefix=spdk_pid1149 [2022-12-22 20:09:33.675721] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-12-22 20:09:33.803886] app.c: 705:spdk_app_start: *NOTICE*: Total cores available: 2
[2022-12-22 20:09:33.832280] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 1
[2022-12-22 20:09:33.832280] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-12-22 20:09:33.834657] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
Mounting filesystem on bdev Nvme0n1 to path /home/miracle/work/task6/bitcask/db/...
done.
[2022-12-22 20:09:33.985402] blobfs_fuse.c: 239:fuse_loop_new_thread: *NOTICE*: Start to loop blobfs on bdev Nvme0n1 mounted at /home/miracle/work/task6/bitcask/db/
```

### 运行测试程序



```
root@cs-exp-zns:/home/miracle/work/task6/bitcask# make run
create file hint.bin successful!
the file filelog.bin open failure or maybe not exist!
create file filelog.bin successful!
value1
please open a database first

value1
key2 does not exist!
```



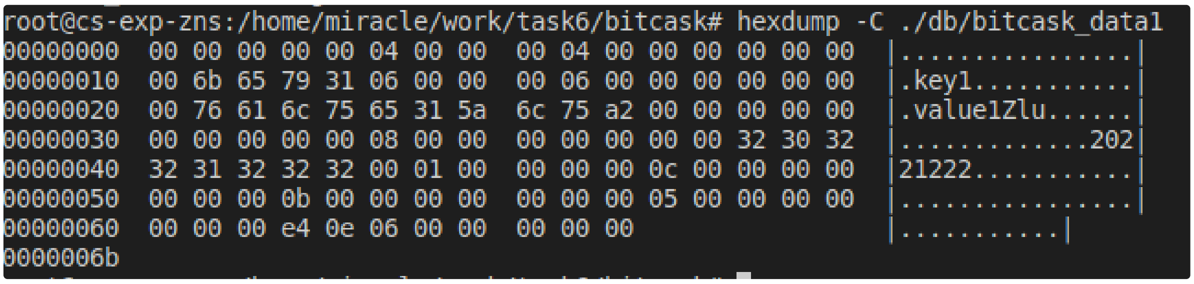
```cpp
int main()
{
    bitcask *db = new bitcask;
    db->Open(db_path);

    db->Put("key1", "value1");
    cout << db->Get("key1") << endl;
    db->Close();
    cout << db->Get("key1") << endl;
    db->Open(db_path);
    cout << db->Get("key1") << endl;
    cout << db->Get("key2") << endl;
    return 0;
}
```

对比测试流程，可以看到

1. 先打开了数据库，程序检测到该处没有任何文件于是进行初始化，并提示hint及filelog文件创建成功
2. `Put("Key1", "value1")`
3. `Read("Key1")` 此时输出了对应的值 `value1`
4. 关闭数据库
5. 再次读取 `Key1` 的值（ `Read("Key1")` ），提示数据库未打开
6. 重新打开数据库
7. 分别获取 `Key1` ， `Key2` 的值
8. 可以看到获取 `Key1` 值时输出了正确的 `value1` ，获取 `Key2` 值时提示 `Key2` 不存在

经过以上测试通过了Open,Close,Put,Get等基本语法要求

**检查二进制存储文件**



```
root@cs-exp-zns:/home/miracle/work/task6/bitcask# hexdump -C ./db/bitcask_data1
00000000  00 00 00 00 00 04 00 00  00 04 00 00 00 00 00 00  |................|
00000010  00 6b 65 79 31 06 00 00  00 06 00 00 00 00 00 00  |.key1...........|
00000020  00 76 61 6c 75 65 31 5a  6c 75 a2 00 00 00 00 00  |.value1Zlu......|
00000030  00 00 00 00 00 08 00 00  00 00 00 00 00 32 30 32  |.............202|
00000040  32 31 32 32 32 00 01 00  00 00 00 0c 00 00 00 00  |21222...........|
00000050  00 00 00 0b 00 00 00 00  00 00 00 05 00 00 00 00  |................|
00000060  00 00 00 e4 0e 06 00 00  00 00 00              |...........|
0000006b
```

可以看出符合之前设计的存储结构

## 实验结论和心得体会

本次实验学习了BlobFS相关特性并在其上实现了KV数据库的Open,Close,Put,Get操作。