**实验六 综合设计实验**

**一、实验目的**

1. 基于SPDK，完成一个综合性设计实验，理解底层NVMe设备驱动到应用程序之间I/O栈。
2. 掌握文件系统或KV数据库原理并进行实现。

**二、实验内容**

1. 设计一个兼容POSIX语义的简易文件系统，可在测试程序中完成fopen, fclose, fread, fwrite, fseek。
2. 在BlobFS中实现文件系统内置压缩、加密功能。
3. 基于BlobFS，设计一个key-value数据库，支持open，put，get，close操作。
4. 如果只想及格，可在SPDK搭建成功前提下，分析BlobFS或FTL源码，给出代码分析报告。可重点选择一个函数，从顶向下分析其I/O栈，例如，对于BlobFS的spdk\_file\_write（）函数，层层分析其调用栈，直到底层nvme驱动调用，画出函数调用图。

**三、实验代码及结果**

实验代码：

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;

}

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)

{main.cpp

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;

}

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);

bitcask.cpp

void Put(string key, string value);

void Open(string path);

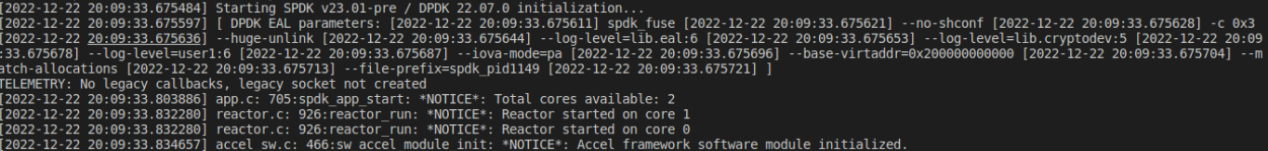
void Close();

~bitcask();

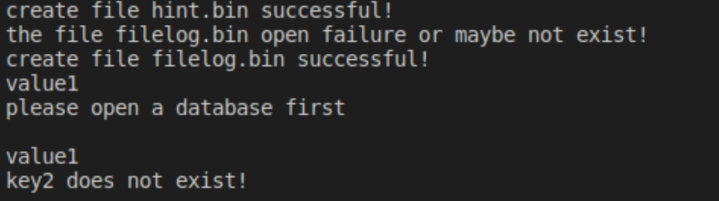
};

#endif \* BITCASK\_H\_ /

挂载blogfs



运行：



首先，打开了数据库，检测到没有任何⽂件于是进⾏初始化，提⽰hint及filelog⽂件创建成功；其次，Put("Key1", "value1")；再输出了对应的值 value1；关闭数据库；再次读取 Key1 的值；重新打开数据库；分别获取 Key1 , Key2 的值；检验完成。

**四、调试和心得体会**

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。理解底层NVMe设备驱动到应用程序之间I/O栈，掌握文件系统或KV数据库原理并进行实现。