实验三 bdev原理和源码分析

**一、实验目的**

1. 学习bdev原理和基本操作接口。

**二、实验内容**

1. 配置bdev运行环境
2. 运行hello\_bdev程序并分析源码
3. 通过bdev接口写入数据并读取

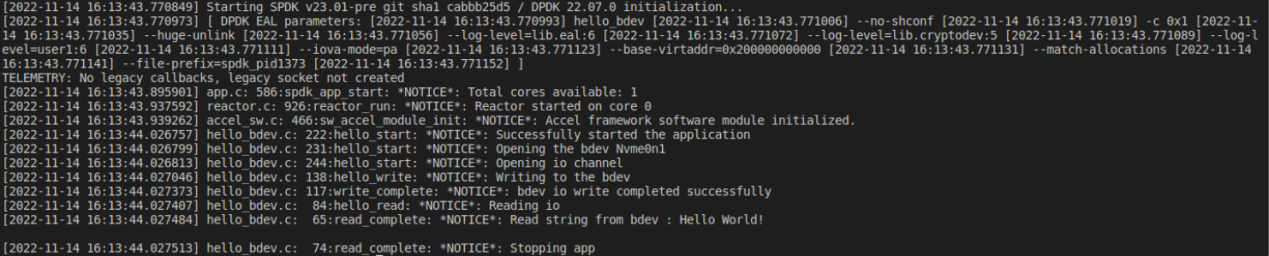
**三、实验代码及结果**

⽣成配置⽂件并运⾏hello\_bdev

./scripts/gen\_nvme.sh - json-with-subsystems > ./build/examples/nvme.json

cd build/examples/

sudo ./hello\_bdev -c nvme.json -b Nvme0n1



代码：

主函数：

main()

int main(int argc, char \*\*argv)

{

struct spdk\_app\_opts opts = {};

int rc = 0;

struct hello\_context\_t hello\_context = {};

/\*Set default values in opts structure. \*/

spdk\_app\_opts\_init(&opts, sizeof(opts));

opts.name = "hello\_bdev";

/\*

\* Parse built-in SPDK command line parameters as well

\* as our custom one(s).

\*/

if ((rc = spdk\_app\_parse\_args(argc, argv, &opts, "b:", NULL,

hello\_bdev\_parse\_arg,

hello\_bdev\_usage)) = SPDK\_APP\_PARSE\_ARGS\_SUCCESS) {

exit(rc);

}

hello\_context.bdev\_name = g\_bdev\_name;

/\*

\* spdk\_app\_start() will initialize the SPDK framework, call

hello\_start(),

\* and then block until spdk\_app\_stop() is called (or if an

initialization

\* error occurs, spdk\_app\_start() will return with rc even without

calling

\* hello\_start().

\*/

rc = spdk\_app\_start(&opts, hello\_start, &hello\_context);

if (rc) {

SPDK\_ERRLOG("ERROR starting application\n");

}

\*

At this point either spdk\_app\_stop() was called, or spdk\_app\_start()

\* failed because of internal error.

/

\*

When the app stops, free up memory that we allocated. /

spdk\_dma\_free(hello\_context.buff);

\*

Gracefully close out all of the SPDK subsystems. /

spdk\_app\_fini();

return rc;

}

Ps:

spdk\_app\_opts\_init() 初始化opts参数

spdk\_app\_parse\_args() 载⼊spdk默认参数和⽤⼾⾃定义参数（如： -c nvme.json 等）

rc = spdk\_app\_start(&opts, hello\_start, &hello\_context) 载⼊SPDK框架，调⽤

hello\_start 函数，并在调⽤ spdk\_app\_stop() 后返回状态值

spdk\_dma\_free() 释放分配的空间

spdk\_app\_fini() 关闭所有SPDK⼦系统

主任务函数：

hello\_start()：

static void hello\_start(void \*arg1)

{

struct hello\_context\_t \*hello\_context = arg1;

uint32\_t buf\_align;

int rc = 0;

hello\_context > bdev = NULL;

hello\_context > bdev\_desc = NULL;

SPDK\_NOTICELOG("Successfully started the application\n");

\*

\* There can be many bdevs configured, but this application will only use

\* the one input by the user at runtime.

\*

\* Open the bdev by calling spdk\_bdev\_open\_ext() with its name.

\* The function will return a descriptor

/

SPDK\_NOTICELOG("Opening the bdev %s\n", hello\_context > bdev\_name);

rc = spdk\_bdev\_open\_ext(hello\_context > bdev\_name, true,

hello\_bdev\_event\_cb, NULL,

&hello\_context > bdev\_desc);

if (rc) {

SPDK\_ERRLOG("Could not open bdev: %s\n", hello\_context > bdev\_name);

spdk\_app\_stop(-1);

return;

}

\*

A bdev pointer is valid while the bdev is opened. /

hello\_context > bdev = spdk\_bdev\_desc\_get\_bdev(hello\_context > bdev\_desc);

SPDK\_NOTICELOG("Opening io channel\n");

\*

Open I/O channel /

hello\_context > bdev\_io\_channel = spdk\_bdev\_get\_io\_channel(hello\_context-

>bdev\_desc);

if (hello\_context > bdev\_io\_channel = NULL) {

SPDK\_ERRLOG("Could not create bdev I/O channel ! \n");

spdk\_bdev\_close(hello\_context > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

\*

Allocate memory for the write buffer.

\* Initialize the write buffer with the string "Hello World!"

/

hello\_context > buff\_size = spdk\_bdev\_get\_block\_size(hello\_context > bdev)

\*

spdk\_bdev\_get\_write\_unit\_size(hello\_context > bdev);

buf\_align = spdk\_bdev\_get\_buf\_align(hello\_context > bdev);

hello\_context > buff = spdk\_dma\_zmalloc(hello\_context > buff\_size,

buf\_align, NULL);

if (!hello\_context > buff) {

SPDK\_ERRLOG("Failed to allocate buffer\n");

spdk\_put\_io\_channel(hello\_context > bdev\_io\_channel);

spdk\_bdev\_close(hello\_context > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

snprintf(hello\_context > buff, hello\_context > buff\_size, "%s", "Hello

World!\n");

if (spdk\_bdev\_is\_zoned(hello\_context > bdev)) {

hello\_reset\_zone(hello\_context);

\*

If bdev is zoned, the callback, reset\_zone\_complete, will call

hello\_write() /

return;

}

hello\_write(hello\_context);

}

Ps:

spdk\_bdev\_open\_ext() 通过设备名（如运⾏程序是附加的参数 -b Nvme0n1 ）打开bdev，返

回⼀个descriptor

spdk\_bdev\_desc\_get\_bdev() 通过descriptor获取bdev指针

spdk\_bdev\_get\_io\_channel() 通过descriptor获取I/O通道

hello\_context->buff\_size= . , hello\_context->buff= . , snprintf( . ) 为写⼊

buffer分配空间并写⼊字符串

hello\_write() 调⽤写⼊函数

写⼊函数:

hello\_write():

static void hello\_write(void \*arg)

{

struct hello\_context\_t \*hello\_context = arg;

int rc = 0;

SPDK\_NOTICELOG("Writing to the bdev\n");

rc = spdk\_bdev\_write(hello\_context > bdev\_desc, hello\_context-

>bdev\_io\_channel,

hello\_context > buff, 0, hello\_context > buff\_size,

write\_complete,

hello\_context);

if (rc = -ENOMEM) {

SPDK\_NOTICELOG("Queueing io\n");

\*

In case we cannot perform I/O now, queue I/O /

hello\_context > bdev\_io\_wait.bdev = hello\_context > bdev;

hello\_context > bdev\_io\_wait.cb\_fn = hello\_write;

0

-EINVAL(-22) -ENOMEM(-12) -EBADF(-9)

rc = spdk\_bdev\_write()

rc == ?

Success

return

offset and/or nbytes are not aligned or out of range

spdk\_put\_io\_channel()

spdk\_bdev\_io buffer cannot be allocated

spdk\_bdev\_queue\_io\_wait()

desc not open for writing

spdk\_bdev\_close()

spdk\_app\_stop(-1)

hello\_context > bdev\_io\_wait.cb\_arg = hello\_context;

spdk\_bdev\_queue\_io\_wait(hello\_context > bdev, hello\_context-

>bdev\_io\_channel,

&hello\_context > bdev\_io\_wait);

} else if (rc) {

SPDK\_ERRLOG("%s error while writing to bdev: %d\n", spdk\_strerror(-

rc), rc);

spdk\_put\_io\_channel(hello\_context > bdev\_io\_channel);

spdk\_bdev\_close(hello\_context > bdev\_desc);

spdk\_app\_stop(-1);

}

}

Ps：

spdk\_bdev\_write() 向bdev写⼊，并在完成后调⽤相应回调函数

spdk\_bdev\_queue\_io\_wait() 加⼊I/O等待队列

spdk\_put\_io\_channel() 释放I/O通道，并在释放最后⼀个通道后调⽤销毁回调函数

spdk\_bdev\_close() 关闭块设备（bdev，block device）

写⼊回调函数:

write\_complete():

static void write\_complete(struct spdk\_bdev\_io \*bdev\_io, bool success, void

\*cb\_arg)

{

struct hello\_context\_t \*hello\_context = cb\_arg;

\*

Complete the I/O /

spdk\_bdev\_free\_io(bdev\_io);

if (success) {

SPDK\_NOTICELOG("bdev io write completed successfully\n");

} else {

SPDK\_ERRLOG("bdev io write error: %d\n", EIO);

spdk\_put\_io\_channel(hello\_context > bdev\_io\_channel);

spdk\_bdev\_close(hello\_context > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

\*

Zero the buffer so that we can use it for reading /

memset(hello\_context > buff, 0, hello\_context > buff\_size);

hello\_read(hello\_context);

}

Ps:

spdk\_bdev\_free\_io() Free an I/O request

memset( . ) 将buffer置为0,便于后续存储读取的数据

hello\_read() 调⽤读取函数

读取函数:

hello\_read() :

static void hello\_read(void \*arg)

{

struct hello\_context\_t \*hello\_context = arg;

int rc = 0;

SPDK\_NOTICELOG("Reading io\n");

rc = spdk\_bdev\_read(hello\_context > bdev\_desc, hello\_context-

>bdev\_io\_channel,

hello\_context > buff, 0, hello\_context > buff\_size,

read\_complete,

hello\_context);

if (rc = -ENOMEM) {

SPDK\_NOTICELOG("Queueing io\n");

\*

In case we cannot perform I/O now, queue I/O /

hello\_context > bdev\_io\_wait.bdev = hello\_context > bdev;

hello\_context > bdev\_io\_wait.cb\_fn = hello\_read;

hello\_context > bdev\_io\_wait.cb\_arg = hello\_context;

spdk\_bdev\_queue\_io\_wait(hello\_context > bdev, hello\_context-

>bdev\_io\_channel,

&hello\_context > bdev\_io\_wait);

} else if (rc) {

SPDK\_ERRLOG("%s error while reading from bdev: %d\n", spdk\_strerror(-

rc), rc);

spdk\_put\_io\_channel(hello\_context > bdev\_io\_channel);

spdk\_bdev\_close(hello\_context > bdev\_desc);

spdk\_app\_stop(-1);

}

}

Ps：

spdk\_bdev\_write() 从bdev读取，并在完成后调⽤相应回调函数

读取回调函数:

read\_complete():  
static void read\_complete(struct spdk\_bdev\_io \*bdev\_io, bool success, void

\*cb\_arg)

{

struct hello\_context\_t \*hello\_context = cb\_arg;

if (success) {

SPDK\_NOTICELOG("Read string from bdev : %s\n", hello\_context > buff);

} else {

SPDK\_ERRLOG("bdev io read error\n");

}

\*

Complete the bdev io and close the channel /

spdk\_bdev\_free\_io(bdev\_io);

spdk\_put\_io\_channel(hello\_context > bdev\_io\_channel);

spdk\_bdev\_close(hello\_context > bdev\_desc);

SPDK\_NOTICELOG("Stopping app\n");

spdk\_app\_stop(success ? 0 : -1);

}

修改hello\_bdev.c，实现⾃定义字符串读写:

miracle\_bdev.c:

#include "spdk/stdinc.h"

#include "spdk/thread.h"

#include "spdk/bdev.h"

#include "spdk/env.h"

#include "spdk/event.h"

#include "spdk/log.h"

#include "spdk/string.h"

#include "spdk/bdev\_zone.h"

#include <math.h>

static char \*g\_bdev\_name = "Nvme0n1";

const int DATA\_LENGTH = 256\*1024;

struct my\_context

{

struct spdk\_bdev \*bdev;

struct spdk\_bdev\_desc \*bdev\_desc;

struct spdk\_io\_channel \*bdev\_io\_channel;

char \*buff;

uint32\_t buff\_size;

char \*bdev\_name;

struct spdk\_bdev\_io\_wait\_entry bdev\_io\_wait;

};

static char \*generate\_str(void)

{

char \*str = (char \*)malloc(DATA\_LENGTH \* 8);

memset(str, 0, DATA\_LENGTH\*8);

if (str)

{

int i;

for (i = 0; i < DATA\_LENGTH; + i)

{

str[i] = '0'+(i%10);

}

return str;

}

else

{

return NULL;

}

}

static void save\_data(const char \*file\_path, char \*str)

{

FILE \*fp = fopen(file\_path, "w");

fprintf(fp, "%s", str);

fclose(fp);

}

static int miracle\_bdev\_parse\_arg(int ch, char \*arg)

{

switch (ch)

{

case 'b':

g\_bdev\_name = arg;

break;

default:

return -EINVAL;

}

return 0;

}

static void miracle\_bdev\_usage(void)

{

printf(" -b <bdev> name of the bdev to use\n");

}

static void bdev\_event\_cb(enum spdk\_bdev\_event\_type type, struct spdk\_bdev

\*bdev, void \*event\_ctx)

{

SPDK\_NOTICELOG("Unsupported bdev event: type %d\n", type);

}

static void read\_complete(struct spdk\_bdev\_io \*bdev\_io, bool success, void

\*cb\_arg)

{

struct my\_context \*p = cb\_arg;

if (success)

{

SPDK\_NOTICELOG("Reading Successfully, Saveing to data.out ! \n");

save\_data("./data.out", p > buff);

}

else

{

SPDK\_ERRLOG("bdev io read error\n");

}

spdk\_bdev\_free\_io(bdev\_io);

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

SPDK\_NOTICELOG("Stopping app\n");

spdk\_app\_stop(success ? 0 : -1);

}

static void start\_read(void \*arg)

{

struct my\_context \*p = arg;

int rc = 0;

SPDK\_NOTICELOG("Reading io\n");

rc = spdk\_bdev\_read(p > bdev\_desc, p > bdev\_io\_channel, p > buff, 0, p-

>buff\_size, read\_complete, p);

if (rc = -ENOMEM)

{

SPDK\_NOTICELOG("Queueing io\n");

p > bdev\_io\_wait.bdev = p > bdev;

p > bdev\_io\_wait.cb\_fn = start\_read;

p > bdev\_io\_wait.cb\_arg = p;

spdk\_bdev\_queue\_io\_wait(p > bdev, p > bdev\_io\_channel, &p-

>bdev\_io\_wait);

}

else if (rc)

{

SPDK\_ERRLOG("%s error while reading from bdev: %d\n", spdk\_strerror(-

rc), rc);

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

}

}

static void write\_complete(struct spdk\_bdev\_io \*bdev\_io, bool success, void

\*cb\_arg)

{

struct my\_context \*p = cb\_arg;

spdk\_bdev\_free\_io(bdev\_io);

if (success)

{

SPDK\_NOTICELOG("bdev io write completed successfully\n");

}

else

{

SPDK\_ERRLOG("bdev io write error: %d\n", EIO);

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

memset(p > buff, 0, p > buff\_size);

start\_read(p);

}

static void start\_write(void \*arg)

{

struct my\_context \*p = arg;

int rc = 0;

SPDK\_NOTICELOG("Writing to the bdev\n");

rc = spdk\_bdev\_write(p > bdev\_desc, p > bdev\_io\_channel, p > buff, 0, p-

>buff\_size, write\_complete, p);

if (rc = -ENOMEM)

{

SPDK\_NOTICELOG("Queueing io\n");

p > bdev\_io\_wait.bdev = p > bdev;

p > bdev\_io\_wait.cb\_fn = start\_write;

p > bdev\_io\_wait.cb\_arg = p;

spdk\_bdev\_queue\_io\_wait(p > bdev, p > bdev\_io\_channel, &p-

>bdev\_io\_wait);

}

else if (rc)

{

SPDK\_ERRLOG("%s error while writing to bdev: %d\n", spdk\_strerror(-

rc), rc);

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

}

}

static void miracle\_bdev(void \*arg)

{

struct my\_context \*p = arg;

uint32\_t buf\_align;

uint32\_t block\_size;

int rc = 0;

p > bdev = NULL;

p > bdev\_desc = NULL;

SPDK\_NOTICELOG("Successfully started the application\n");

SPDK\_NOTICELOG("Opening the bdev %s\n", p > bdev\_name);

rc = spdk\_bdev\_open\_ext(p > bdev\_name, true, bdev\_event\_cb, NULL, &p-

>bdev\_desc);

if (rc)

{

SPDK\_ERRLOG("Could not open bdev: %s\n", p > bdev\_name);

spdk\_app\_stop(-1);

return;

}

p > bdev = spdk\_bdev\_desc\_get\_bdev(p > bdev\_desc);

SPDK\_NOTICELOG("Opening io channel\n");

p > bdev\_io\_channel = spdk\_bdev\_get\_io\_channel(p > bdev\_desc);

if (p > bdev\_io\_channel = NULL)

{

SPDK\_ERRLOG("Could not create bdev I/O channel ! \n");

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

block\_size = spdk\_bdev\_get\_block\_size(p > bdev);

buf\_align = spdk\_bdev\_get\_buf\_align(p > bdev);

p > buff\_size = ceil((double)(DATA\_LENGTH+1)/block\_size)\*block\_size;

p > buff = spdk\_dma\_zmalloc(p > buff\_size, buf\_align, NULL);

if (!p > buff)

{

SPDK\_ERRLOG("Failed to allocate buffer\n");

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

SPDK\_NOTICELOG("Generating Data\n");

char \*str = generate\_str();

if (str){

snprintf(p > buff, p > buff\_size, "%s", str);

free(str);

SPDK\_NOTICELOG("Saving Data to ./data.txt\n");

save\_data("./data.in", p > buff);

start\_write(p);

}

else{

SPDK\_ERRLOG("Could not generate data ! \n");

spdk\_put\_io\_channel(p > bdev\_io\_channel);

spdk\_bdev\_close(p > bdev\_desc);

spdk\_app\_stop(-1);

return;

}

}

int main(int argc, char \*\*argv)

{

struct spdk\_app\_opts opts = {};

int rc = 0;

struct my\_context context = {};

spdk\_app\_opts\_init(&opts, sizeof(opts));

opts.name = "miracle\_bdev";

rc = spdk\_app\_parse\_args(argc, argv, &opts, "b:", NULL,

miracle\_bdev\_parse\_arg, miracle\_bdev\_usage);

if (rc = SPDK\_APP\_PARSE\_ARGS\_SUCCESS)

{

exit(rc);

}

context.bdev\_name = g\_bdev\_name;

rc = spdk\_app\_start(&opts, miracle\_bdev, &context);

if (rc)

{

SPDK\_ERRLOG("ERROR starting applicatoin\n");

}

spdk\_dma\_free(context.buff);

spdk\_app\_fini();

return rc;

}

Makefile :

SPDK\_ROOT\_DIR := /home/miracle/work/task2/spdk

include $(SPDK\_ROOT\_DIR)/mk/spdk.common.mk

include $(SPDK\_ROOT\_DIR)/mk/spdk.modules.mk

APP = miracle\_bdev

C\_SRCS := miracle\_bdev.c

SPDK\_LIB\_LIST = $(ALL\_MODULES\_LIST) event event\_bdev

include $(SPDK\_ROOT\_DIR)/mk/spdk.app.mk

run: all

@ echo "Finished Compiling, Cleaning intermediate files"

@ rm -f miracle\_bdev.d miracle\_bdev.o

@ echo "Generating bdev-config"

@ $(SPDK\_ROOT\_DIR)/scripts/gen\_nvme.sh - json-with-subsystems >

./miracle\_bdev.json

@ echo "Generated bdev-config, Runing Program"

@ sudo ./miracle\_bdev -c ./miracle\_bdev.json

@ echo "Comparing Writing Data and Reading Data"

@ echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Size \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

@ du -h data.\*

@ echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

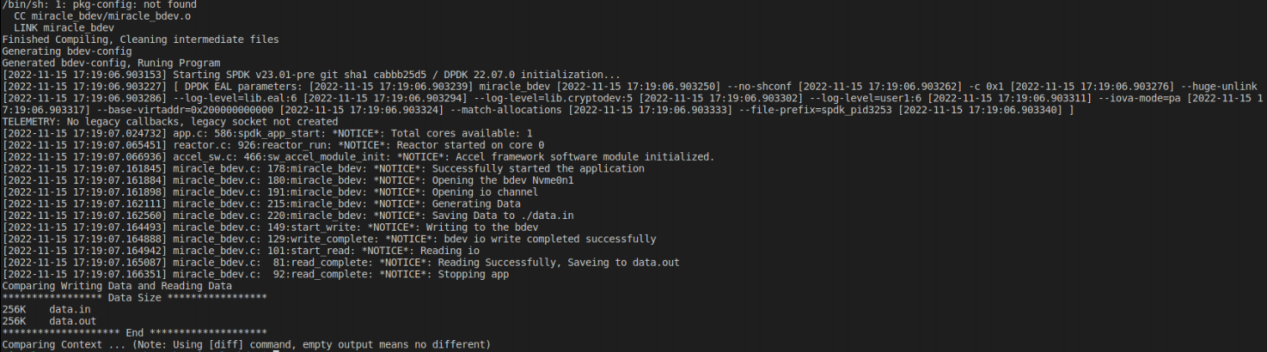
@ echo "Comparing Context . (Note: Using [diff] command, empty output

means no different)"

@ diff data.in data.out

运行：

Make run



**四、调试和心得体会**

本次实验配置了bdev运行环境，运行hello\_bdev程序并分析源码，并通过bdev接口写入数据并读取。