实验四 blobstore原理和源码分析

**一、实验目的**

1. 掌握blobstore原理
2. 完成基于nvme的blobstore读写。

**二、实验内容**

1. 学习Blob基本原理
2. 完成hello\_blob 程序运行
3. 修改底层bdev为nvme
4. **实验代码及结果**

Makefile:

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

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

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

APP = miracle\_blob

C\_SRCS := miracle\_blob.c

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

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

run: all

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

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

./miracle\_bdev.json

@ sudo ./miracle\_blob ./miracle\_bdev.json

miracle\_blob.c:

#include "spdk/stdinc.h"

#include "spdk/bdev.h"

#include "spdk/env.h"

#include "spdk/event.h"

#include "spdk/blob\_bdev.h"

#include "spdk/blob.h"

#include "spdk/log.h"

#include "spdk/string.h"

struct my\_context

{

struct spdk\_blob\_store \*bs;

struct spdk\_blob \*blob;

spdk\_blob\_id blobid;

struct spdk\_io\_channel \*channel;

uint8\_t \*read\_buff;

uint8\_t \*write\_buff;

uint64\_t io\_unit\_size;

int rc;

};

static void cleanup(struct my\_context \*p)

{

spdk\_free(p > read\_buff);

spdk\_free(p > write\_buff);

free(p);

}

static void unload\_complete(void \*cb\_arg, int bserrno)

{

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

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

SPDK\_ERRLOG("Error %d unloading the bobstore\n", bserrno);

p > rc = bserrno;

}

spdk\_app\_stop(p > rc);

}

static void unload\_bs(struct my\_context \*p, char \*msg, int bserrno)

{

if (bserrno)

{

SPDK\_ERRLOG("%s (err %d)\n", msg, bserrno);

p > rc = bserrno;

}

if (p > bs)

{

if (p > channel)

{

spdk\_bs\_free\_io\_channel(p > channel);

}

spdk\_bs\_unload(p > bs, unload\_complete, p);

}

else

{

spdk\_app\_stop(bserrno);

}

}

static void delete\_complete(void \*arg1, int bserrno)

{

struct my\_context \*p = arg1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in delete completion", bserrno);

return;

}

unload\_bs(p, "", 0);

}

static void delete\_blob(void \*arg1, int bserrno)

{

struct my\_context \*p = arg1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in close completion", bserrno);

return;

}

spdk\_bs\_delete\_blob(p > bs, p > blobid, delete\_complete, p);

}

static void read\_complete(void \*arg1, int bserrno)

{

struct my\_context \*p = arg1;

int match\_res = -1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in read completion", bserrno);

return;

}

match\_res = memcmp(p > write\_buff, p > read\_buff, p > io\_unit\_size);

if (match\_res)

{

unload\_bs(p, "Error in data compare", -1);

return;

}

else

{

SPDK\_NOTICELOG("read SUCCESS and data matches!\n");

}

spdk\_blob\_close(p > blob, delete\_blob, p);

}

static void read\_blob(struct my\_context \*p)

{

SPDK\_NOTICELOG("entry\n");

p > read\_buff = spdk\_malloc(p > io\_unit\_size, 0x1000, NULL,

SPDK\_ENV\_LCORE\_ID\_ANY, SPDK\_MALLOC\_DMA);

if (p > read\_buff = NULL)

{

unload\_bs(p, "Error in memory allocation", -ENOMEM);

return;

}

spdk\_blob\_io\_read(p > blob, p > channel, p > read\_buff, 0, 1, read\_complete,

p);

}

static void write\_complete(void \*arg1, int bserrno)

{

struct my\_context \*p = arg1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in write completion", bserrno);

return;

}

read\_blob(p);

}

static void blob\_write(struct my\_context \*p)

{

SPDK\_NOTICELOG("entry\n");

p > write\_buff = spdk\_malloc(p > io\_unit\_size, 0x1000, NULL,

SPDK\_ENV\_LCORE\_ID\_ANY, SPDK\_MALLOC\_DMA);

if (p > write\_buff = NULL)

{

unload\_bs(p, "Error in allocating memory", -ENOMEM);

return;

}

memset(p > write\_buff, 0x5a, p > io\_unit\_size);

p > channel = spdk\_bs\_alloc\_io\_channel(p > bs);

if (p > channel = NULL)

{

unload\_bs(p, "Error in allocating channel", -ENOMEM);

return;

}

spdk\_blob\_io\_write(p > blob, p > channel, p > write\_buff, 0, 1,

write\_complete, p);

}

static void sync\_complete(void \*arg1, int bserrno)

{

struct my\_context \*p = arg1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in sync callback", bserrno);

return;

}

blob\_write(p);

}

static void resize\_complete(void \*cb\_arg, int bserrno)

{

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

uint64\_t total = 0;

if (bserrno)

{

unload\_bs(p, "Error in blob resize", bserrno);

return;

}

total = spdk\_blob\_get\_num\_clusters(p > blob);

SPDK\_NOTICELOG("resized blob now has USED clusters of %" PRIu64 "\n",

total);

spdk\_blob\_sync\_md(p > blob, sync\_complete, p);

}

static void open\_complete(void \*cb\_arg, struct spdk\_blob \*blob, int bserrno)

{

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

uint64\_t free = 0;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in open completion", bserrno);

return;

}

p > blob = blob;

free = spdk\_bs\_free\_cluster\_count(p > bs);

SPDK\_NOTICELOG("blobstore has FREE clusters of %" PRIu64 "\n", free);

spdk\_blob\_resize(p > blob, free, resize\_complete, p);

}

static void blob\_create\_complete(void \*arg1, spdk\_blob\_id blobid, int

bserrno)

{

struct my\_context \*p = arg1;

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error in blob create callback", bserrno);

return;

}

p > blobid = blobid;

SPDK\_NOTICELOG("new blob id %" PRIu64 "\n", p > blobid);

spdk\_bs\_open\_blob(p > bs, p > blobid, open\_complete, p);

}

static void create\_blob(struct my\_context \*p)

{

SPDK\_NOTICELOG("entry\n");

spdk\_bs\_create\_blob(p > bs, blob\_create\_complete, p);

}

static void bs\_init\_complete(void \*cb\_arg, struct spdk\_blob\_store \*bs, int

bserrno)

{

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

SPDK\_NOTICELOG("entry\n");

if (bserrno)

{

unload\_bs(p, "Error initing the blobstore", bserrno);

return;

}

p > bs = bs;

SPDK\_NOTICELOG("blobstore: %p\n", p > bs);

p > io\_unit\_size = spdk\_bs\_get\_io\_unit\_size(p > bs);

create\_blob(p);

}

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

spdk\_bdev \*bdev, void \*event\_ctx)

{

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

}

static void hello\_start(void \*arg1)

{

struct my\_context \*p = arg1;

struct spdk\_bs\_dev \*bs\_dev = NULL;

int rc;

rc = spdk\_bdev\_create\_bs\_dev\_ext("Nvme0n1", base\_bdev\_event\_cb, NULL,

&bs\_dev);

if (rc = 0)

{

SPDK\_ERRLOG("Could not create blob bdev, %s ! \n", spdk\_strerror(-

rc));

spdk\_app\_stop(-1);

return;

}

spdk\_bs\_init(bs\_dev, NULL, bs\_init\_complete, p);

}

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

{

struct spdk\_app\_opts opts = {};

int rc = 0;

struct my\_context \*p = NULL;

SPDK\_NOTICELOG("entry\n");

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

opts.name = "hello\_miracle";

opts.json\_config\_file = argv[1];

p = calloc(1, sizeof(struct my\_context));

if (p)

{

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

if (rc)

{

SPDK\_NOTICELOG("ERROR!\n");

}

else

{

SPDK\_NOTICELOG("SUCCESS!\n");

}

cleanup(p);

}

else

{

SPDK\_ERRLOG("Could not alloc hello\_context struct ! \n");

rc = -ENOMEM;

}

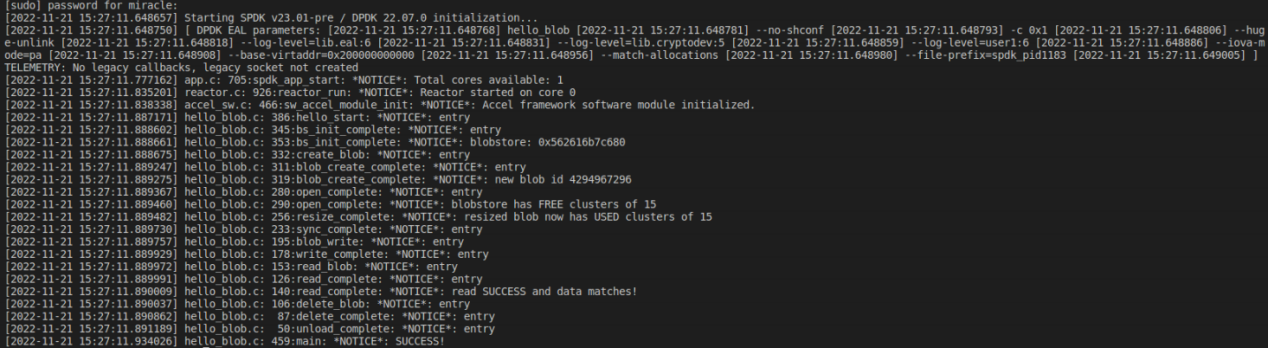
spdk\_app\_fini();

return rc;

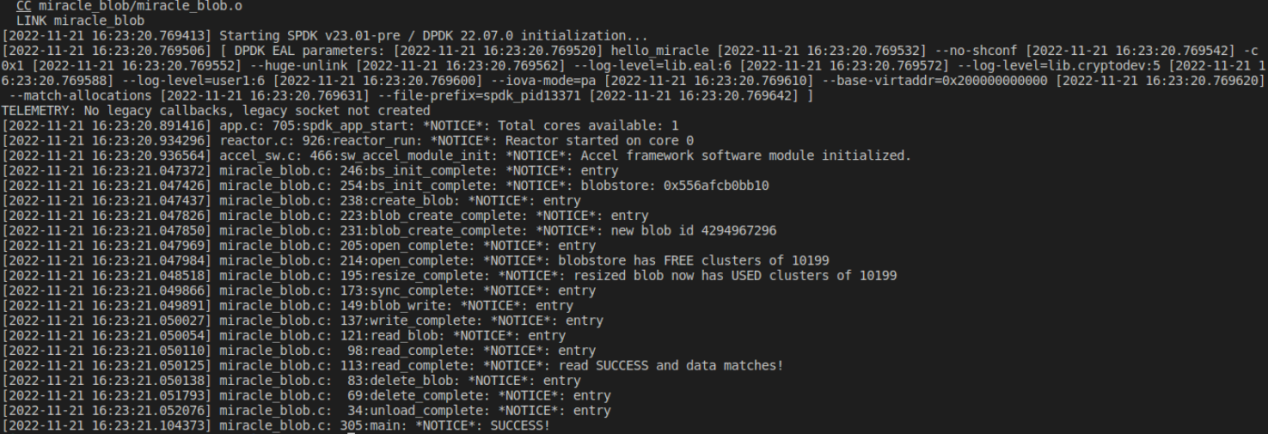
}

运⾏hello\_blob:

sudo ./build/examples/hello\_blob ./examples/blob/hello\_world/hello\_blob.json



运行：make run:



**四、调试和心得体会**

本次实验学习了Blob基本原理，完成了hello\_bdev程序运行，并最修改底层bdev为nvme。