实验四、blobstore原理和源码分析

实验目的

- 学习blob原理和基本接口操作

实验内容

- 学习Blob基本原理
- 完成hello_blob 程序运行
- 修改底层bdev为nvme

实验过程和步骤

运行hello_blob示例

启动虚拟机

```
./start.sh ssd
```

初始化环境

```
sudo scripts/setup.sh
```

运行hello_blob

在spdk文件夹下

```
sudo ./build/examples/hello_blob ./examples/blob/hello_world/hello_blob.json
```

```
miracle@cs-exp-zns:~/work/spdk$ sudo ./build/examples/hello_blob ./examples/blob/hello_world/hello_blob.json
[sudo] password for miracle:
[2022-11-21 15:27:11.648657] Starting SPDK v23.01-pre / DPDK 22.07.0 initialization...
[2022-11-21 15:27:11.648750] [ DPDK EAL parameters: [2022-11-21 15:27:11.648768] hello_blob [2022-11-21 15:27:11.648781] --no-shconf [2022-11-21 15:27:11.648793] -c 0x1 [2022-11-21 15:27:11.648806] --hug
e-unlink [2022-11-21 15:27:11.648818] --log-level=lib.eal:6 [2022-11-21 15:27:11.648831] --log-level=lib.cryptodev:5 [2022-11-21 15:27:11.648859] --log-level=user1:6 [2022-11-21 15:27:11.648886] --iova-m
ode=pa [2022-11-21 15:27:11.648908] --base-virtaddr=0x200000000000 [2022-11-21 15:27:11.648956] --match-allocations [2022-11-21 15:27:11.648980] --file-prefix=spdk_pid1183 [2022-11-21 15:27:11.649005] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-11-21 15:27:11.777162] app.c: 705:spdk_app_start: *NOTICE*: Total cores available: 1
[2022-11-21 15:27:11.835201] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-11-21 15:27:11.838338] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
[2022-11-21 15:27:11.887171] hello_blob.c: 306:hello_start: *NOTICE*: entry
[2022-11-21 15:27:11.888602] hello_blob.c: 345:bs_init_complete: *NOTICE*: entry
[2022-11-21 15:27:11.888661] hello_blob.c: 353:bs_init_complete: *NOTICE*: blobstore: 0x562616b7c680
[2022-11-21 15:27:11.888675] hello_blob.c: 332:create_blob: *NOTICE*: entry
[2022-11-21 15:27:11.889247] hello_blob.c: 311:blob_create_complete: *NOTICE*: entry
[2022-11-21 15:27:11.889275] hello_blob.c: 319:blob_create_complete: *NOTICE*: new blob id 4294967296
[2022-11-21 15:27:11.889367] hello_blob.c: 280:open_complete: *NOTICE*: entry
[2022-11-21 15:27:11.889460] hello_blob.c: 290:open_complete: *NOTICE*: blobstore has FREE clusters of 15
[2022-11-21 15:27:11.889482] hello_blob.c: 256:resize_complete: *NOTICE*: resized blob now has USED clusters of 15
[2022-11-21 15:27:11.889730] hello_blob.c: 233:sync_complete: *NOTICE*: entry
[2022-11-21 15:27:11.889757] hello_blob.c: 195:blob_write: *NOTICE*: entry
[2022-11-21 15:27:11.889939] hello_blob.c: 178:write_complete: *NOTICE*: entry
[2022-11-21 15:27:11.889972] hello_blob.c: 153:read_blob: *NOTICE*: entry
[2022-11-21 15:27:11.889991] hello_blob.c: 126:read_complete: *NOTICE*: entry
[2022-11-21 15:27:11.890009] hello_blob.c: 140:read_complete: *NOTICE*: read SUCCESS and data matches!
[2022-11-21 15:27:11.890037] hello_blob.c: 106:delete_blob: *NOTICE*: entry
[2022-11-21 15:27:11.890062] hello_blob.c:  87:delete_complete: *NOTICE*: entry
[2022-11-21 15:27:11.891189] hello_blob.c:  50:unload_complete: *NOTICE*: entry
[2022-11-21 15:27:11.934026] hello_blob.c: 459:main: *NOTICE*: SUCCESS!
miracle@cs-exp-zns:~/work/spdk$
```

修改hello_blob.c，替换ramdisk为nvme

`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;
}
```

`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
```

运行结果

```
miracle@cs-exp-zns:~/work/task4/miracle_blob$ make run
  CC miracle_blob/miracle_blob.o
  LINK miracle_blob
[2022-11-21 16:23:20.769413] Starting SPDK v23.01-pre / DPDK 22.07.0 initialization...
[2022-11-21 16:23:20.769506] [ DPDK EAL parameters: [2022-11-21 16:23:20.769520] hello_miracle [2022-11-21 16:23:20.769532] --no-shconf [2022-11-21 16:23:20.769542] -c
0x1 [2022-11-21 16:23:20.769552] --huge-unlink [2022-11-21 16:23:20.769562] --log-level=lib.eal:6 [2022-11-21 16:23:20.769572] --log-level=lib.cryptodev:5 [2022-11-21 1
6:23:20.769588] --log-level=user1:6 [2022-11-21 16:23:20.769600] --iova-mode=pa [2022-11-21 16:23:20.769610] --base-virtaddr=0x200000000000 [2022-11-21 16:23:20.769620]
 --match-allocations [2022-11-21 16:23:20.769631] --file-prefix=spdk_pid13371 [2022-11-21 16:23:20.769642] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-11-21 16:23:20.891416] app.c: 705:spdk_app_start: *NOTICE*: Total cores available: 1
[2022-11-21 16:23:20.934296] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-11-21 16:23:20.936564] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
[2022-11-21 16:23:21.047372] miracle_blob.c: 246:bs_init_complete: *NOTICE*: entry
[2022-11-21 16:23:21.047426] miracle_blob.c: 254:bs_init_complete: *NOTICE*: blobstore: 0x556afcb0bb10
[2022-11-21 16:23:21.047437] miracle_blob.c: 238:create_blob: *NOTICE*: entry
[2022-11-21 16:23:21.047826] miracle_blob.c: 223:blob_create_complete: *NOTICE*: entry
[2022-11-21 16:23:21.047850] miracle_blob.c: 231:blob_create_complete: *NOTICE*: new blob id 4294967296
[2022-11-21 16:23:21.047969] miracle_blob.c: 205:open_complete: *NOTICE*: entry
[2022-11-21 16:23:21.047984] miracle_blob.c: 214:open_complete: *NOTICE*: blobstore has FREE clusters of 10199
[2022-11-21 16:23:21.048518] miracle_blob.c: 195:resize_complete: *NOTICE*: resized blob now has USED clusters of 10199
[2022-11-21 16:23:21.049866] miracle_blob.c: 173:sync_complete: *NOTICE*: entry
[2022-11-21 16:23:21.049891] miracle_blob.c: 149:blob_write: *NOTICE*: entry
[2022-11-21 16:23:21.050027] miracle_blob.c: 137:write_complete: *NOTICE*: entry
[2022-11-21 16:23:21.050054] miracle_blob.c: 121:read_blob: *NOTICE*: entry
[2022-11-21 16:23:21.050110] miracle_blob.c:  98:read_complete: *NOTICE*: entry
[2022-11-21 16:23:21.050125] miracle_blob.c: 113:read_complete: *NOTICE*: read SUCCESS and data matches!
[2022-11-21 16:23:21.050138] miracle_blob.c:  83:delete_blob: *NOTICE*: entry
[2022-11-21 16:23:21.051793] miracle_blob.c:  69:delete_complete: *NOTICE*: entry
[2022-11-21 16:23:21.052076] miracle_blob.c:  34:unload_complete: *NOTICE*: entry
[2022-11-21 16:23:21.104373] miracle_blob.c: 305:main: *NOTICE*: SUCCESS!
miracle@cs-exp-zns:~/work/task4/miracle_blob$
```

实验结论和心得体会

本次实验学习了Blob基本原理，运行并分析了hello_bdev程序，并最修改底层bdev为nvme并成功运行。