

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

三、实验代码及结果

1. hello_blob运行

同前面实验一样，启用NVME环境，初始化

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

运行 hello_blob

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

```
zhp@zhp:~/spdk$ sudo ./build/examples/hello_blob ./examples/blob/hello_world/hello_blob.json
[2022-12-13 07:58:08.972678] Starting SPDK v23.01-pre git sha1 a64acd100 / DPDK 22.07.0 initialization...
[2022-12-13 07:58:08.977573] [ DPDK EAL parameters: [2022-12-13 07:58:08.977790] hello_blob [2022-12-13 07:58:08.977888] --no-shconf [2022-12-13 07:58:08.977944] -c 0x1 [2022-12-13 07:58:08.977992] --huge-unlink [2022-12-13 07:58:08.978037] --log-level=lib.eal:6 [2022-12-13 07:58:08.978080] --log-level=lib.cryptodev:5 [2022-12-13 07:58:08.978122] --log-level=user1:6 [2022-12-13 07:58:08.978167] --iova-mode=pa [2022-12-13 07:58:08.978215] --base-virtaddr=0x20000000000 [2022-12-13 07:58:08.978259] --match-allocations [2022-12-13 07:58:08.978301] --file-prefix=spdk_pid1214 [2022-12-13 07:58:08.978354] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-12-13 07:58:09.225898] app.c: 705:spdk_app_start: *NOTICE*: Total cores available: 1
[2022-12-13 07:58:09.387712] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-12-13 07:58:09.429297] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
[2022-12-13 07:58:09.677869] hello_blob.c: 386:hello_start: *NOTICE*: entry
[2022-12-13 07:58:09.685645] hello_blob.c: 345:bs_init_complete: *NOTICE*: entry
[2022-12-13 07:58:09.685943] hello_blob.c: 353:bs_init_complete: *NOTICE*: blobstore: 0x56552869db60
[2022-12-13 07:58:09.686053] hello_blob.c: 332:create_blob: *NOTICE*: entry
[2022-12-13 07:58:09.692529] hello_blob.c: 311:blob_create_complete: *NOTICE*: entry
[2022-12-13 07:58:09.692708] hello_blob.c: 319:blob_create_complete: *NOTICE*: new blob id 4294967296
[2022-12-13 07:58:09.693386] hello_blob.c: 280:open_complete: *NOTICE*: entry
[2022-12-13 07:58:09.693573] hello_blob.c: 290:open_complete: *NOTICE*: blobstore has FREE clusters of 15
[2022-12-13 07:58:09.694307] hello_blob.c: 256:resize_complete: *NOTICE*: resized blob now has USED clusters of 15
[2022-12-13 07:58:09.695928] hello_blob.c: 233:sync_complete: *NOTICE*: entry
[2022-12-13 07:58:09.696098] hello_blob.c: 195:blob_write: *NOTICE*: entry
[2022-12-13 07:58:09.696565] hello_blob.c: 178:write_complete: *NOTICE*: entry
[2022-12-13 07:58:09.696667] hello_blob.c: 153:read_blob: *NOTICE*: entry
[2022-12-13 07:58:09.696820] hello_blob.c: 126:read_complete: *NOTICE*: entry
[2022-12-13 07:58:09.697024] hello_blob.c: 140:read_complete: *NOTICE*: read SUCCESS and data matches!
[2022-12-13 07:58:09.697241] hello_blob.c: 106:delete_blob: *NOTICE*: entry
[2022-12-13 07:58:09.701013] hello_blob.c: 87:delete_complete: *NOTICE*: entry
[2022-12-13 07:58:09.701747] hello_blob.c: 50:unload_complete: *NOTICE*: entry
[2022-12-13 07:58:09.885919] hello_blob.c: 459:main: *NOTICE*: SUCCESS!
```

2.修改底层bdev为nvme

hello_blob 的代码逻辑和 hello_bdev 差不多，除了多了 sync_complete 同步检测 resize_complete 等几个类似调整文件的内置函数。

将 hello_start 中的 Malloc0 替换成 NvmeOn1

```

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);
}

```

整体源码如下:

```

#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 bsrno)
{
    struct my_context *p = cb_arg;

    SPDK_NOTICELOG("entry\n");
    if (bserrno)
    {
        SPDK_ERRLOG("Error %d unloading the bobstore\n", bsrno);
        p->rc = bsrno;
    }
}

```

```

        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 berrno)
{
    struct my_context *p = cb_arg;
    uint64_t free = 0;

    SPDK_NOTICELOG("entry\n");
    if (berrno)
    {
        unload_bs(p, "Error in open completion", berrno);
        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 berrno)
{
    struct my_context *p = arg1;

    SPDK_NOTICELOG("entry\n");
    if (berrno)
    {
        unload_bs(p, "Error in blob create callback", berrno);
        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 berrno)
{
    struct my_context *p = cb_arg;

    SPDK_NOTICELOG("entry\n");
    if (berrno)
    {
        unload_bs(p, "Error initing the blobstore", berrno);
        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 = "zhp_blob";
    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文件:

```

zhp@zhp: ~/spdk/task4

SPDK_ROOT_DIR := /home/zhp/spdk
include $(SPDK_ROOT_DIR)/mk/spdk.common.mk
include $(SPDK_ROOT_DIR)/mk/spdk.modules.mk

APP = zhp_blob

C_SRCS := zhp_blob.c

SPDK_LIB_LIST = $(ALL_MODULES_LIST) event event_bdev

include $(SPDK_ROOT_DIR)/mk/spdk.app.mk

run:all
@ $(SPDK_ROOT_DIR)/scripts/gen_nvme.sh --json-with-subsystems > ./zhp_bdev.json
@ sudo ./zhp_blob ./zhp_bdev.json

```

得到运行结果

```

zhp@zhp:~/spdk/task4$ make run
/bin/sh: 1: [: not found
/bin/sh: 1: pkg-config: not found
CC task4/zhp_blob.o
LINK zhp_blob
[2022-12-13 08:10:17.390124] Starting SPDK v23.01-pre git sha1 a64acd100 / DDPK 22.07.0 initialization...
[2022-12-13 08:10:17.394249] [ DDPK EAL parameters: [2022-12-13 08:10:17.394424] zhp_blob [2022-12-13 08:10:17.394522] --no-shconf [2022-12-13 08:10:17.394575] -c 0x1 [2022-12-13 08:10:17.394622] --huge-unlink [2022-12-13 08:10:17.394665] --log-level=lib.eal:6 [2022-12-13 08:10:17.394710] --log-level=lib.cryptodev:5 [2022-12-13 08:10:17.394753] --log-level=user1:6 [2022-12-13 08:10:17.394797] --iova-mode=pa [2022-12-13 08:10:17.394842] --base-virtaddr=0x20000000000 [2022-12-13 08:10:17.394885] --match-allocations [2022-12-13 08:10:17.394927] --file-prefix=spdk_pid1273 [2022-12-13 08:10:17.394981] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-12-13 08:10:17.673201] app.c: 705:spdk_app_start: *NOTICE*: Total cores available: 1
[2022-12-13 08:10:17.817112] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-12-13 08:10:17.828654] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
[2022-12-13 08:10:18.087130] zhp_blob.c: 246:bs_init complete: *NOTICE*: entry
[2022-12-13 08:10:18.093357] zhp_blob.c: 254:bs_init complete: *NOTICE*: blobstore: 0x5649172cf9a0
[2022-12-13 08:10:18.093464] zhp_blob.c: 238:create_blob: *NOTICE*: entry
[2022-12-13 08:10:18.094887] zhp_blob.c: 223:blob_create_complete: *NOTICE*: entry
[2022-12-13 08:10:18.095038] zhp_blob.c: 231:blob_create_complete: *NOTICE*: new blob id 4294967296
[2022-12-13 08:10:18.102718] zhp_blob.c: 205:open_complete: *NOTICE*: entry
[2022-12-13 08:10:18.103075] zhp_blob.c: 214:open_complete: *NOTICE*: blobstore has FREE clusters of 10199
[2022-12-13 08:10:18.107294] zhp_blob.c: 195:resize_complete: *NOTICE*: resized blob now has USED clusters of 10199
[2022-12-13 08:10:18.114924] zhp_blob.c: 173:sync_complete: *NOTICE*: entry
[2022-12-13 08:10:18.119514] zhp_blob.c: 149:blob_write: *NOTICE*: entry
[2022-12-13 08:10:18.121010] zhp_blob.c: 137:write_complete: *NOTICE*: entry
[2022-12-13 08:10:18.121122] zhp_blob.c: 121:read_blob: *NOTICE*: entry
[2022-12-13 08:10:18.121747] zhp_blob.c: 98:read_complete: *NOTICE*: entry
[2022-12-13 08:10:18.121999] zhp_blob.c: 113:read_complete: *NOTICE*: read SUCCESS and data matches!
[2022-12-13 08:10:18.122194] zhp_blob.c: 83:delete_blob: *NOTICE*: entry
[2022-12-13 08:10:18.134680] zhp_blob.c: 69:delete_complete: *NOTICE*: entry
[2022-12-13 08:10:18.154775] zhp_blob.c: 34:unload_complete: *NOTICE*: entry
[2022-12-13 08:10:18.343505] zhp_blob.c: 305:main: *NOTICE*: SUCCESS!
zhp@zhp:~/spdk/task4$

```

可以看到生成的json配置文件与实验三的一样，即Blob是bdev的更高级的封装。

```

zhp@zhp:~/spdk/task4$ cat zhp_bdev.json
zhp@zhp:~/spdk/task4$ ls
Makefile  zhp_bdev.json  zhp_blob  zhp_blob.c  zhp_blob.d  zhp_blob.o
zhp@zhp:~/spdk/task4$

```



```
zhp@zhp: ~/spdk/task4
{
  "subsystems": [
  {
    "subsystem": "bdev",
    "config": [
    {
      "method": "bdev_nvme_attach_controller",
      "params": {
        "trtype": "PCIe",
        "name": "Nvme0",
        "traddr": "0000:00:04.0"
      }
    }
  ]
}
```

四、调试和心得体会

pBlob构建在bdev之上，是对数据存储的进一步抽象，类似于文件，但是不具备文件POSIX接口，可近似按文件形式理解。