

实验三、bdev原理和源码分析

实验目的

- 学习bdev原理和基本接口操作

实验内容

- 配置bdev运行环境
- 运行hello_bdev程序并分析源码
- 通过bdev接口写入数据并读取

实验过程和步骤

运行hello_bdev示例

启动虚拟机

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

初始化环境

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

生成配置文件

```
./scripts/gen_nvme.sh --json-with-subsystems > ./build/examples/nvme.json
```

运行hello_bdev

```
cd build/examples/  
sudo ./hello_bdev -c nvme.json -b Nvme0n1
```

```
miracle@cs-exp-zns:~/work/task2/spdk/build/examples$ sudo ./hello_bdev -c nvme.json -b Nvme0n1  
[2022-11-14 16:13:43.770849] Starting SPDK v23.01-pre git sha1 cabbb25d5 / DPDK 22.07.0 initialization...  
[2022-11-14 16:13:43.770973] [ DPDK EAL parameters: [2022-11-14 16:13:43.770993] hello_bdev [2022-11-14 16:13:43.771006] --no-shconf [2022-11-14 16:13:43.771019] -c 0x1 [2022-11-14 16:13:43.771035] --huge-unlink [2022-11-14 16:13:43.771056] --log-level=lib.eal:6 [2022-11-14 16:13:43.771072] --log-level=lib.cryptodev:5 [2022-11-14 16:13:43.771089] --log-level=user1:6 [2022-11-14 16:13:43.771111] --iova-mode=pa [2022-11-14 16:13:43.771123] --base-virtaddr=0x200000000000 [2022-11-14 16:13:43.771131] --match-allocations [2022-11-14 16:13:43.771141] --file-prefix=spdk pid1373 [2022-11-14 16:13:43.771152] ]  
TELEMETRY: No legacy callbacks, legacy socket not created  
[2022-11-14 16:13:43.895901] app.c: 586:spdk_app_start: *NOTICE*: Total cores available: 1  
[2022-11-14 16:13:43.937592] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0  
[2022-11-14 16:13:43.939262] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.  
[2022-11-14 16:13:44.026757] hello_bdev.c: 222:hello_start: *NOTICE*: Successfully started the application  
[2022-11-14 16:13:44.026799] hello_bdev.c: 231:hello_start: *NOTICE*: Opening the bdev Nvme0n1  
[2022-11-14 16:13:44.026813] hello_bdev.c: 244:hello_start: *NOTICE*: Opening io channel  
[2022-11-14 16:13:44.027046] hello_bdev.c: 138:hello_write: *NOTICE*: Writing to the bdev  
[2022-11-14 16:13:44.027273] hello_bdev.c: 117:write_complete: *NOTICE*: bdev io write completed successfully  
[2022-11-14 16:13:44.027407] hello_bdev.c: 84:hello_read: *NOTICE*: Reading io  
[2022-11-14 16:13:44.027484] hello_bdev.c: 65:read_complete: *NOTICE*: Read string from bdev : Hello World!  
[2022-11-14 16:13:44.027513] hello_bdev.c: 74:read_complete: *NOTICE*: Stopping app
```

分析hello_bdev.c源码

main() 主函数

```
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;

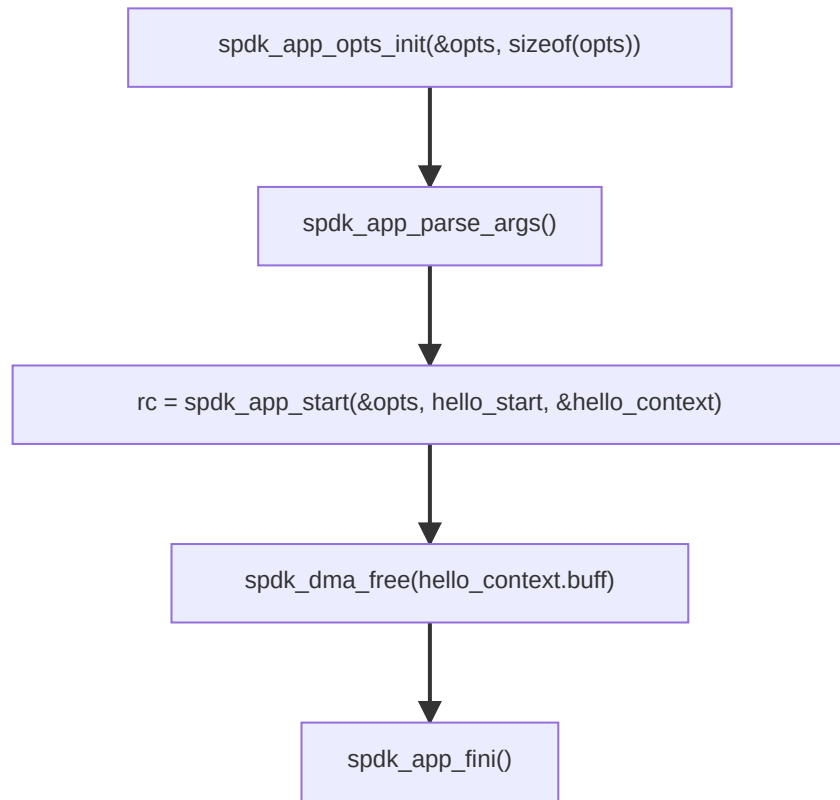
    /*
     * spdk_app_start() will initialize the SPDK framework, call
     hello_start(),
     * and then block until spdk_app_stop() is called (or if an
     initialization
     * error occurs, spdk_app_start() will return with rc even without
     calling
     * hello_start().
     */
    rc = spdk_app_start(&opts, 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;
}
```

主要流程为



其中：

- `spdk_app_opts_init()` 初始化opts参数
- `spdk_app_parse_args()` 载入spdk默认参数和用户自定义参数（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

`hello_start()` 主任务函数

```
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;
    }

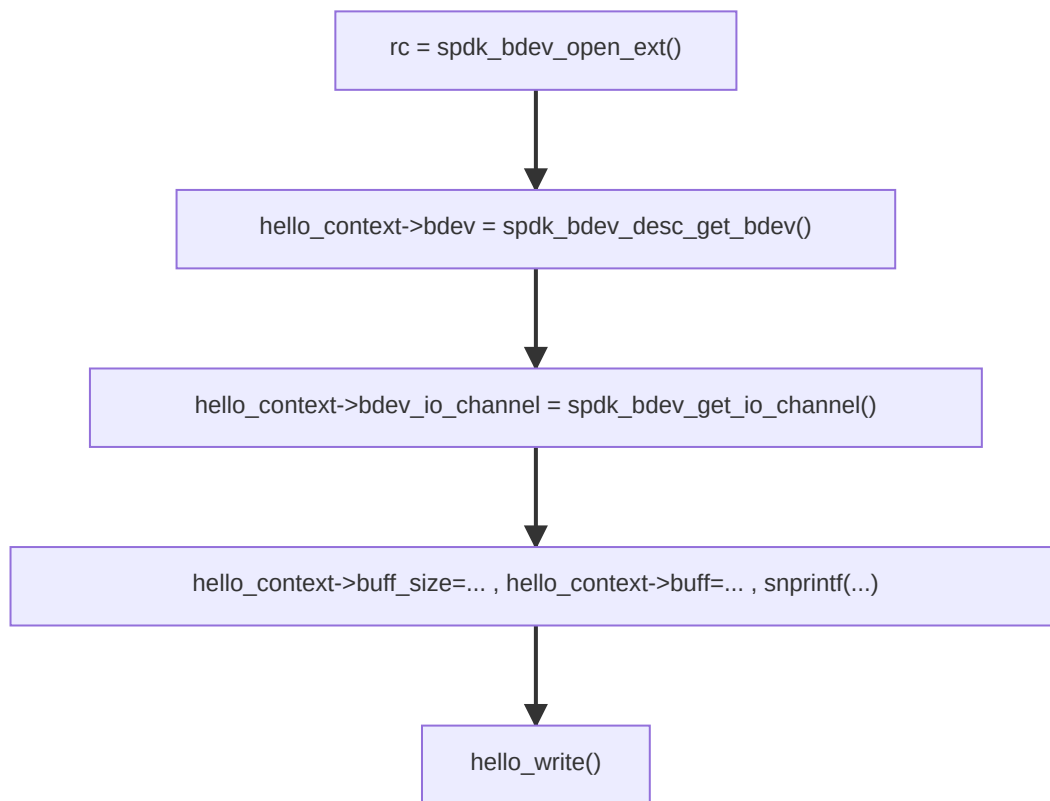
    /* Allocate memory for the write buffer.
     * Initialize the write buffer with the string "Hello World!"
     */
    hello_context→buff_size = spdk_bdev_get_block_size(hello_context→bdev)
*
        spdk_bdev_get_write_unit_size(hello_context→bdev);
    buf_align = spdk_bdev_get_buf_align(hello_context→bdev);
    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);
}

```

主要流程为



其中：

- `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

`hello_write()` 写入函数

下面对写入函数进行分析

```
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);

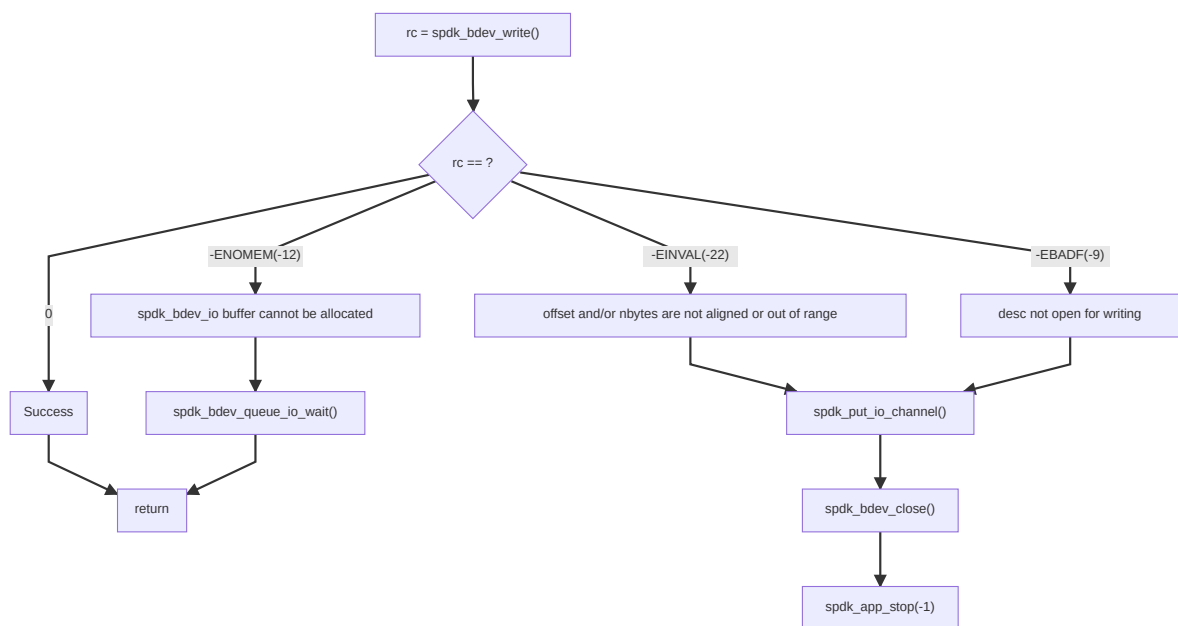
    if (rc == -ENOMEM) {
        SPDK_NOTICELOG("Queueing io\n");
        /* In case we cannot perform I/O now, queue I/O */
        hello_context->bdev_io_wait.bdev = hello_context->bdev;
        hello_context->bdev_io_wait.cb_fn = hello_write;
    }
}
```

```

        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);
    }
}
}

```

主要流程为



其中：

- `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)

write_complete() 写入回调函数

```
static void write_complete(struct spdk_bdev_io *bdev_io, bool success, void
*cb_arg)
{
    struct hello_context_t *hello_context = cb_arg;

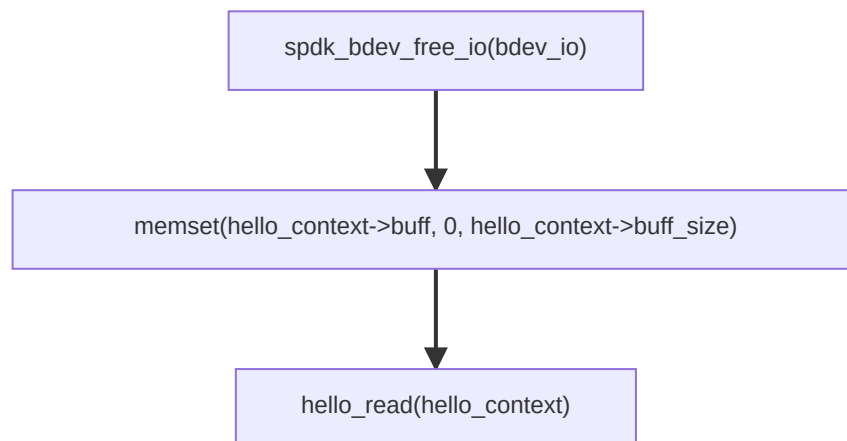
    /* Complete the I/O */
    spdk_bdev_free_io(bdev_io);

    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);
}
```

主要流程为



其中：

- `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

hello_read() 读取函数

```
static void hello_read(void *arg)
{
    struct hello_context_t *hello_context = arg;
    int rc = 0;
```

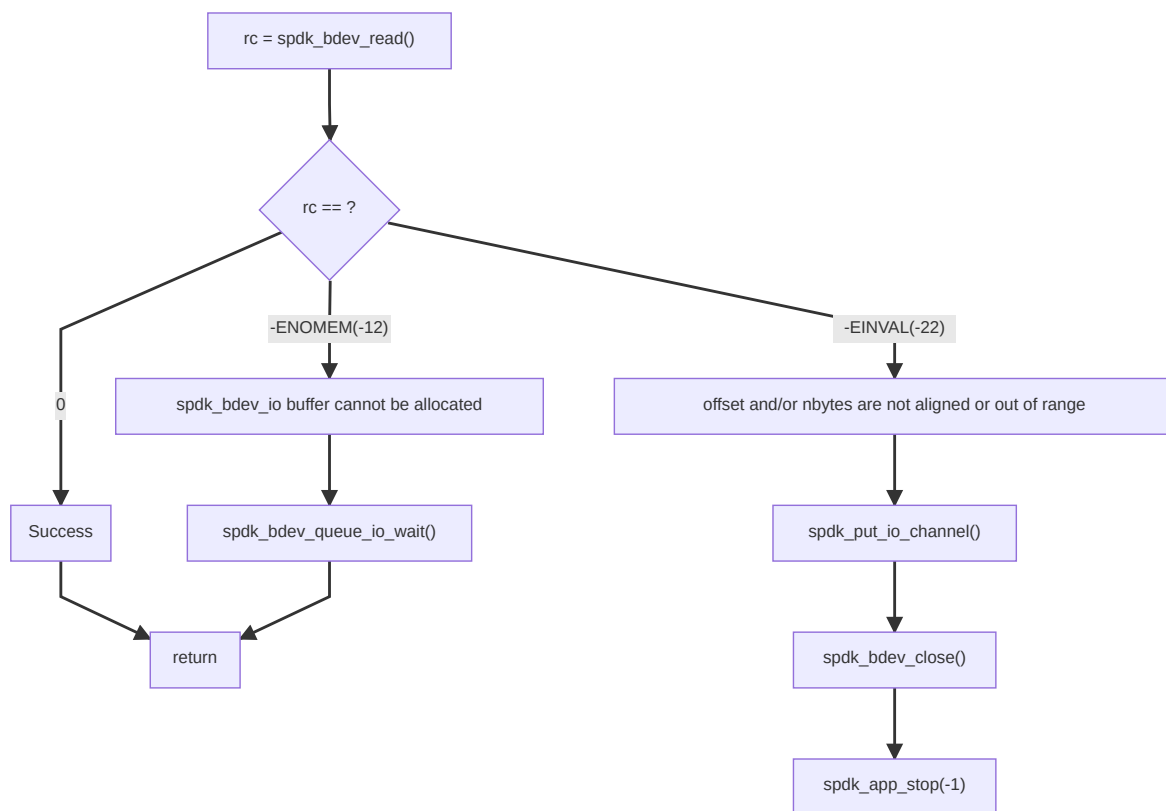
```

        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);
        }
    }
}

```

主要流程为



其中：

- `spdk_bdev_write()` 从bdev读取，并在完成后调用相应回调函数

`read_complete()` 读取回调函数

```

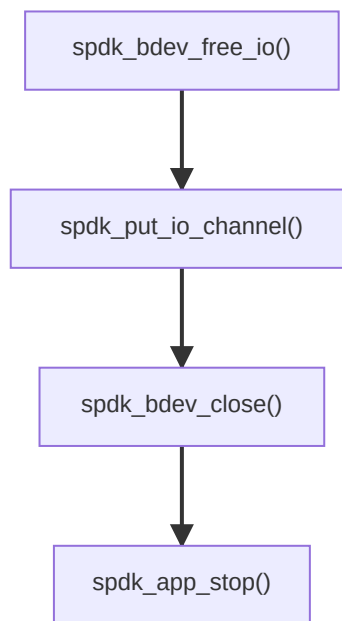
static void read_complete(struct spdk_bdev_io *bdev_io, bool success, void
*cb_arg)
{
    struct hello_context_t *hello_context = cb_arg;

    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);
}
  
```

```
}
```

主要流程为



修改hello_bdev.c，实现自定义字符串读写

生成256KB字符串数据，修改hello_bdev.c源码将字符串数据通过bdev写入，之后再读取，验证结果是否正确

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
```

运行结果

```
miracle@ecs-esp-zns:~/work/task3/miracle_bdev$ make run
/bin/sh: 1: pkg-config: not found
cc miracle_bdev/miracle_bdev.o
lnkx miracle_bdev
Finished Compiling, Cleaning intermediate files
Generating bdev-config
Generated bdev-config, Running Program
[2022-11-15 17:19:06.903153] Starting SPDK v23.01-pre git sha1 cabb25d5 / DPDK 22.07.0 initialization...
[2022-11-15 17:19:06.903227] [ DPDK EAL parameters: [2022-11-15 17:19:06.903239] miracle_bdev [2022-11-15 17:19:06.903250] --no-shconf [2022-11-15 17:19:06.903262] -c 0x1 [2022-11-15 17:19:06.903276] --huge-unlink
[2022-11-15 17:19:06.903286] --log-level=lib.eal:6 [2022-11-15 17:19:06.903294] --log-level=lib.cryptodev:5 [2022-11-15 17:19:06.903302] --log-level=usr:1:6 [2022-11-15 17:19:06.903311] --iova-mode=pa [2022-11-15 1
7:19:06.903317] --base-virtaddr=0x200000000000 [2022-11-15 17:19:06.903324] --match-allocations [2022-11-15 17:19:06.903333] --file-prefix=spdk_pid3253 [2022-11-15 17:19:06.903340] ]
TELEMETRY: No legacy callbacks, legacy socket not created
[2022-11-15 17:19:07.024732] app.c: 586:spdk_app_start: *NOTICE*: Reactor started on core 0
[2022-11-15 17:19:07.065451] reactor.c: 926:reactor_run: *NOTICE*: Reactor started on core 0
[2022-11-15 17:19:07.066936] accel_sw.c: 466:sw_accel_module_init: *NOTICE*: Accel framework software module initialized.
[2022-11-15 17:19:07.161845] miracle_bdev.c: 178:miracle_bdev: *NOTICE*: Successfully started the application
[2022-11-15 17:19:07.161884] miracle_bdev.c: 180:miracle_bdev: *NOTICE*: Opening the bdev Nvme0n1
[2022-11-15 17:19:07.161898] miracle_bdev.c: 191:miracle_bdev: *NOTICE*: Opening io channel
[2022-11-15 17:19:07.162111] miracle_bdev.c: 215:miracle_bdev: *NOTICE*: Generating Data
[2022-11-15 17:19:07.162550] miracle_bdev.c: 228:miracle_bdev: *NOTICE*: Saving Data to ./data.in
[2022-11-15 17:19:07.164493] miracle_bdev.c: 149:start write: *NOTICE*: Writing to the bdev
[2022-11-15 17:19:07.164880] miracle_bdev.c: 129:write complete: *NOTICE*: bdev io write completed successfully
[2022-11-15 17:19:07.164942] miracle_bdev.c: 181:start read: *NOTICE*: Reading io
[2022-11-15 17:19:07.165807] miracle_bdev.c: 81:read complete: *NOTICE*: Reading Successfully, Saving to data.out
[2022-11-15 17:19:07.166351] miracle_bdev.c: 92:read complete: *NOTICE*: Stopping app
Comparing Writing Data and Reading Data
***** Data Size *****
256K data.in
256K data.out
***** End *****
Comparing Context ... (Note: Using [diff] command, empty output means no different)
miracle@ecs-esp-zns:~/work/task3/miracle_bdev$
```

实验结论和心得体会

本次实验配置了bdev运行环境，运行并分析了hello_bdev程序，并最终独立编写实现了通过bdev接口的数据写入与读取。