임베디드시스템실습 PA1

2017313107 이승태

- kvssd의 qemu코드를 작성하고 user.c를 통해 key-value방법을 확인해 보았습니다.

1, 구조체, 변수 선언

(1) DEV...의 변수로 device의 상태를 확인할 수 있게 했습니다.

(2) CMD..의 변수로 command입력을 받을 수 있게 했습니다.

(3) BLK_NUM으로 최대 블락 넘버를 조절할 수 있게 하고, GC_BLK_NUM은 GC_BLK_NUM 이하로 unused block이 존재하게 된다면, garbage collection을 진행하게 했습니다.

(4) key_pos는 kvssd_key_pos와 비슷한 변수로 s->kvssd_key_pos으로 생각해도 무방합니다.

(5) struct에 키의 존재 여부를 return하는 exist, 할당된 블록을 저장하고 있는 blk, free_blk의 개수를 저장하고 있는 free_blk등으로 구성되었습니다.

```
#define DEV_READY   0
#define DEV_BUSY    1
#define DEV_WAIT    2

#define CMD_PUT     3
#define CMD_GET     4
#define CMD_ERASE   5
#define CMD_EXIST   6

#define BLK_NUM     1234
#define GC_BLK_NUM  1

int key_pos;

enum {
    /* CMD Registers */

    KVSSD_STATUS_REG    =0x00,
    KVSSD_CMD_REG       =0x04,
    KVSSD_KEY_REG       =0x08,
    KVSSD_VALUE_REG     =0x0C,
};

struct goldfish_kvssd_state {
    struct goldfish_device dev;
    int kvssd_fd_data;
    int kvssd_fd_meta;
    uint32_t status; //Ready: 0, Busy: 1
    uint32_t cmd;   //Read: 0, Write: 1
    uint16_t kvssd_key_pos;
    uint16_t kvssd_value_pos;
    uint32_t kvssd_key[4];
    uint32_t kvssd_value[1024];
    uint8_t exist;
    uint32_t blk;
    uint32_t free_blk;
};
```

2. meta, disk파일의 구성

kvssd.meta파일은

key (16 bytes) + 현재 블록의 상태 (‘I’nvalid, ‘U’nused, ‘V’alid) + ‘\n’로 구성되며

16 + 1 + 1 = 18 bytes를 읽으면, 한 블록에 관한 정보를 얻을 수 있습니다.

위에서 차례대로 실제 disk에 쓰여진 block과 매핑됩니다.

kvssd.disk파일은 값을 저장하고 있는 파일입니다.

3. 다음으로는 reg 값을 읽을 때 호출되는 goldfish_kvssd_read입니다.
(1) STATUS_REG를 읽을 때에는 현재 status값을 반환합니다.
(2) KEY_REG를 읽을 경우, 현재 이 키의 블록 값이 exist인지를 return해줍니다.
(3) VALUE_REG를 읽을 경우, 현재 struct에 저장된 value값을 return해줍니다.

```
static uint32_t goldfish_kvssd_read(void* opaque, hwaddr offset)
{
    struct goldfish_kvssd_state* s = (struct goldfish_kvssd_state*)opaque;
    uint32_t temp;

    if ( offset < 0 ) {
        cpu_abort(cpu_single_env, "kvssd_dev_read: Bad offset %" HWADDR_PRIx "\n", offset);
        return 0;
    }

    switch (offset) {
        case KVSSD_STATUS_REG:
            return s->status;
        case KVSSD_KEY_REG:
            return s->exist;
        case KVSSD_VALUE_REG:
            temp = s->kvssd_value_pos;
            s->kvssd_value_pos = (s->kvssd_value_pos+1) % 1024;
            return s->kvssd_value[temp];
    };

    return 0;
}
```

4. garbage_collection
garbage_collection이 시작되었을 때 gc start!를 shell창에 쓰고
메타 파일에서 모든 Invalid block을 찾고,
Invalid block (-> 'I')를 Unused block (-> 'U')로 바꾸어줍니다.

```
void garbage_collection(struct goldfish_kvssd_state *s) {
    printf("gc start!\n");

    char buffer[19];
    int buffer_offset = 16 + 1 + 1;
    lseek(s->kvssd_fd_meta, 0, SEEK_SET);

    while(read(s->kvssd_fd_meta, buffer, buffer_offset))
    {
        if(buffer[16] == 'I')
        {
            buffer[16] = 'U';
            lseek(s->kvssd_fd_meta, -buffer_offset, SEEK_CUR);
            write(s->kvssd_fd_meta, buffer, buffer_offset);
            s->free_blk++;
        }
    }
    return;
}
```

5. new_block_allocate는 새로운 unused block을 할당해주는 함수입니다.
CMD_PUT에 의해서만 불립니다.

```c
void new_block_allocate(struct goldfish_kvssd_state *s) {
    char buffer_end[3];
    char buffer[19];
    int ind = 0;
    int buffer_offset = 16 + 1 + 1;
    lseek(s->kvssd_fd_meta, 0, SEEK_SET);
    while(read(s->kvssd_fd_meta, buffer, buffer_offset))
    {
        if(buffer[16] == 'U')
        {
            lseek(s->kvssd_fd_meta, -buffer_offset, SEEK_CUR);
            write(s->kvssd_fd_meta, s->kvssd_key, 16);
            buffer_end[0] = 'V';
            buffer_end[1] = '\n';
            write(s->kvssd_fd_meta, buffer_end, 2);
            s->blk = ind;
            s->free_blk--;
            return;
        }
        ind++;
    }

    write(s->kvssd_fd_meta, s->kvssd_key, 16);
    buffer_end[0] = 'V';
    buffer_end[1] = '\n';
    write(s->kvssd_fd_meta, buffer_end, 2);
    s->blk = ind;
    s->free_blk--;
    return;
}
```

6. check_key는 valid한 block의 key의 존재 여부를 return해주는 함수로 meta파일을 읽으면서 key의 일치를 판단합니다. 이 함수는 CMD에 따라 각각 다른 일을 수행합니다.

PUT이나 ERASE CMD가 들어오게 되면, 기존의 Valid한 block을 Invalid로 바꾸고 지웠다는 것을 표시하기 위해 return 1을 해줍니다.

또한 GET이 들어오게 되면, 현재 같은 key값의 block number를 써줍니다.

```c
static int check_key(struct goldfish_kvssd_state *s, int what_cmd)
{
    char buffer[19];
    int buffer_offset = 16 + 1 + 1;     // key, valid, enter
    int ind = 0;
    lseek(s->kvssd_fd_meta, 0, SEEK_SET);
    while(read(s->kvssd_fd_meta, buffer, buffer_offset))
    {
        if(buffer[16] == 'V')
        {
            uint32_t compare_num[4];
            memcpy(compare_num, buffer, 16);
            int i;
            for(i=0;i<4;i++)
            {
                if(compare_num[i] != s->kvssd_key[i])   break;
            }
            if(i == 4)
            {
                if(what_cmd == CMD_PUT || what_cmd == CMD_ERASE)
                {
                    buffer[16] = 'I';
                    lseek(s->kvssd_fd_meta, -buffer_offset, SEEK_CUR);
                    write(s->kvssd_fd_meta, buffer, buffer_offset);
                }
                else if(what_cmd == CMD_GET)    s->blk = ind;

                s->exist = 1;
                return 1;
            }
        }
        ind++;
    }
    s->exist = 0;
    return 0;
}
```

7. goldfish_kvssd_data_read, goldfish_kvssd_data_write는 s->blk를 check_key나 new_block_allocate에서 할당받아 value를 받아오거나 써줍니다.

```
static int goldfish_kvssd_data_read(struct goldfish_kvssd_state *s) {
    pread(s->kvssd_fd_data, s->kvssd_value, sizeof(s->kvssd_value), s->blk << PAGE_SHIFT);

    return 0;
}

static int goldfish_kvssd_data_write(struct goldfish_kvssd_state *s) {
    pwrite(s->kvssd_fd_data, s->kvssd_value, sizeof(s->kvssd_value), s->blk << PAGE_SHIFT);
    fsync(s->kvssd_fd_data);

    return 0;
}
```

8. (goldfish_kvssd_read 내부) STATUS에 값을 쓸 경우, key값을 다시 확인해주는 작업이기 때문에 이에 맞는 protocol을 정의해주었습니다.

```
case KVSSD_STATUS_REG:
    if(s->status == DEV_WAIT) {
        if(value == s->kvssd_key[key_pos]) {
            if(key_pos == 3) {
                s->status = DEV_READY;
                key_pos = 0;
            }
            else key_pos = (key_pos+1) % 4;
        }
        else {
            key_pos = 0;
            printf("error, different key\n");
        }
    }
    else    key_pos = 0;
    break;
```

9. CMD_REG에 put값을 넣었을 경우입니다. free_blk이 GC_BLK_NUM보다 작거나 같아지면, gc를 진행합니다. 더 이상 free_blk이 없을 경우 cpu_abort를 일으킵니다.

그 후 put의 protocol을 구현해주었습니다.

(1) key가 있으면 invalid시켜줍니다. (check_key)

(2) 새로운 블록을 할당하고 메타파일을 고칩니다.(new_block_allocate)

(3) 그 후 값을 data파일에 써줍니다.(goldfish_kvssd_data_write)

(4) 마지막으로 인터럽트를 날려주고 status값을 ready로 고쳐줍니다.

```
case CMD_PUT:
    if(s->free_blk <= GC_BLK_NUM)   garbage_collection(s);
    if(s->free_blk == 0)
    {
        printf("We don't have any space to put data!\n");
        cpu_abort(cpu_single_env, "kvssd_cmd: PUT error\n");
        return;
    }
    else
    {
        check_key(s, CMD_PUT);                      // Check key and invalid block
        new_block_allocate(s);                      // Allocate new block and change meta data
        status = goldfish_kvssd_data_write(s);      // Write
        if ( status == 0 ) {
            goldfish_device_set_irq(&s->dev, 0, 1);
            s->status = DEV_READY;
        } else {
            cpu_abort(cpu_single_env, "kvssd_cmd: PUT error\n");
            return;
        }
    }
    break;
```

10. GET의 프로토콜입니다.
(1) key를 확인하여 없으면 Get command not accepted: Wrong key를 반환하고 존재한다면 blk값을 지정해줍니다. (check_key)
(2) 있다면 read로 s->kvssd_value에 저장해줍니다.(goldfish_kvssd_data_read)
(3) 마지막으로 인터럽트를 날려주고 status값을 wait로 고쳐줍니다.

```
case CMD_GET:
    if(check_key(s, CMD_GET))
        status = goldfish_kvssd_data_read(s);
    else    printf("Get command not accepted: Wrong key\n");

    if ( status == 0 ) {
        goldfish_device_set_irq(&s->dev, 0, 1);
        s->status = DEV_WAIT;
    } else {
        cpu_abort(cpu_single_env, "kvssd_cmd: GET error\n");
        return;
    }
    break;
```

11. EXIST의 protocol입니다.
(1) key를 확인해주고 s->exist값을 있으면 1을 없으면 0으로 세팅해줍니다. (check_key)
(2) 마지막으로 인터럽트를 날려주고 status값을 wait로 고쳐줍니다.

```
case CMD_EXIST:
    check_key(s, CMD_EXIST);
    goldfish_device_set_irq(&s->dev, 0, 1);
    s->status = DEV_WAIT;
    break;
```

12. ERASE의 protocol입니다.
(1) key를 확인해주고 존재한다면 meta파일에서 Invalid를 진행합니다. (check_key)
(2) 마지막으로 인터럽트를 날려주고 status값을 ready로 고쳐줍니다.

```
case CMD_ERASE:
    if(!check_key(s, CMD_ERASE))
        printf("There is no key to erase!\n");
    goldfish_device_set_irq(&s->dev, 0, 1);
    s->status = DEV_READY;
    break;
```

13. struct의 초기값 지정을 해주는 goldfish_kvssd_init함수입니다.
다른 컴퓨터를 사용할 때에는 open의 경로 값을 바꿔주어야 합니다.
맨 아래에는 처음 init을 할 때 free_blk의 개수를 세어줍니다.
meta파일에 저장되어있지 않기 때문에, 직접 계산하여 기록해줍니다.

```
void goldfish_kvssd_init(void)
{
    struct goldfish_kvssd_state *s;

    s = (struct goldfish_kvssd_state *)g_malloc0(sizeof(*s));
    s->kvssd_key_pos = 0;
    s->kvssd_value_pos = 0;
    s->dev.name = "goldfish_kvssd";
    s->dev.base = 0;
    s->dev.size = 0x1000;
    s->dev.irq_count = 1;
    s->dev.irq = 15;
    //Change to absolute path & Add authority(0777)
    s->kvssd_fd_data = open("/home/lee/emu-2.2-release/external/qemu/kvssd.disk", O_RDWR | O_CREAT, 0777);
    s->kvssd_fd_meta = open("/home/lee/emu-2.2-release/external/qemu/kvssd.meta", O_RDWR | O_CREAT, 0777);
    s->free_blk = BLK_NUM;
    key_pos = 0;

    char buffer[19];
    int buffer_offset = 16 + 1 + 1;
    lseek(s->kvssd_fd_meta, 0, SEEK_SET);
    while(read(s->kvssd_fd_meta, buffer, buffer_offset))
    {
        if(buffer[16] != 'U')
            s->free_blk--;
    }
    if(s->free_blk <= GC_BLK_NUM)   garbage_collection(s);
```

14. user.c (key_maker, value_maker)

위의 작동을 실험해보기 위해 user.c를 작성했습니다.

key_maker값은 들어온 a값을 랜덤적으로 4개의 키에 집어넣습니다.(a값이 들어가거나 들어가지 않거나)

value_maker는 들어온 a값으로 모든 value값을 초기화 시켜줍니다.

```
void ioctl_key_maker(struct ioctl_env *e, int a)
{
    int i;

    ind = ((ind + 7) % 15) + 1;

    int indtmp = ind;
    for(i=0;i<4;i++)
    {
        if(indtmp & 1)      e->key[i] = a;
        else                e->key[i] = 0;
        indtmp >>= 1;
    }
}

void ioctl_value_maker(struct ioctl_env *e, int a)
{
    int i;
    for(i=0;i<1024;i++) e->value[i] = 0;
    for(i=0;i<1024;i++)
    {
        int j;
        int atmp = a;
        for(j=0;j<4;j++)
        {
            e->value[i] |= atmp;
            atmp <<= 8;
        }
    }
}
```

15. 실험 1

key값을 만들고 (1~80값이 랜덤적으로 4개의 key값에 들어가므로(key가 다 들어가지 않는 경우는 제외함) (80 * (2^4-1)) = 1200) 1200개의 key값이 들어가 있으므로 아무리 put을 해도 cpu_abort가 출력될 일은 없을 것이다. 그러므로 15000번의 put을 하고, 랜덤적으로 key

값 존재를 확인하고 있으면 읽고, 지우고 다시 확인하는 작업을 50번 진행하였다.

```
for(i=0;i<15000;i++)
{
    ioctl_key_maker(&env, rand() % 80 + 1);
    ioctl_value_maker(&env, (i%20) + 'a');
    ioctl(fd, CMD_PUT, &env);   //pass address of env to kernel
}
printf("\n");
printf("PUT number: %d\n", i);

//exist
for(i=1;i<50;i++)
{
    ioctl_key_maker(&env, i);
    printf("key:%d %d %d %d\n", env.key[0], env.key[1], env.key[2], env.key[3]);
    if(ioctl(fd, CMD_EXIST, &env))
    {
        printf("exist\n");
        ioctl(fd, CMD_GET, &env);
        printf("data env: %c\n", env.value[100]);

        ioctl(fd, CMD_ERASE, &env);

        if(ioctl(fd, CMD_EXIST, &env))
            printf("%d exist\n", i);
        else
            printf("%d doesn't exist\n", i);
        printf("\n");
    }
    else
        printf("doesn't exist\n\n");
```

다음은 실험 결과이다. 성공적으로 15000번의 put이 일어났고, key값의 존재 여부를 물어보고, 있으면 읽고 data를 출력하고, 지운 후 다시 존재 여부를 확인했더니 거의 모든 키가 존재했다.

```
lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 push user /data/local/tmp
user: 1 file pushed, 0 skipped. 62.9 MB/s (7776 bytes in 0.000s)
lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 shell
generic_x86_64:/ $ su
generic_x86_64:/ # cd data/local/tmp
generic_x86_64:/data/local/tmp # ./user

PUT number: 15000
key:1 0 0 1
exist
data env: a
1 doesn't exist

key:0 2 0 0
exist
data env: b
2 doesn't exist

key:0 3 0 3
exist
data env: c
3 doesn't exist

key:4 4 0 0
exist
data env: n
4 doesn't exist

key:5 5 0 5
exist
data env: o
5 doesn't exist
```

16. 실험 2

이보다 더 많은 100 * 15 = 1500개의 key를 생성해서 3000번의 put을 진행시켰더니 cpu_abort가 발생한 모습이다.

```
for(i=0;i<3000;i++)
{
    ioctl_key_maker(&env, rand() % 100 + 1);
    ioctl_value_maker(&env, (i%20) + 'a');
    ioctl(fd, CMD_PUT, &env);   //pass address of env to kernel
}
printf("\n");
printf("PUT number: %d\n", i);

//exist
for(i=1;i<50;i++)
{
    ioctl_key_maker(&env, i);
    printf("key:%d %d %d %d\n", env.key[0], env.key[1], env.key[2], env.key[3]);
    if(ioctl(fd, CMD_EXIST, &env))
    {
        printf("exist\n");
        ioctl(fd, CMD_GET, &env);
        printf("data env: %c\n", env.value[100]);

        ioctl(fd, CMD_ERASE, &env);

        if(ioctl(fd, CMD_EXIST, &env))
            printf("%d exist\n", i);
        else
            printf("%d doesn't exist\n", i);
        printf("\n");
    }
    else
        printf("doesn't exist\n\n");
```

```
            close(fd);
=000000          ^
=000000 3 warnings generated.
 FTW=00 lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 push user /data/local/tmp
R1=0000 adb: error: failed to get feature set: device 'emulator-5556' not found
R3=0000 lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 push user /data/local/tmp
R5=0000 adb: error: failed to get feature set: device offline
R7=0000 lee@lee-GF75-Thin-9SC:~$
0000000 lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 push user /data/local/tmp
c2d543a user: 1 file pushed, 0 skipped. 49.2 MB/s (7776 bytes in 0.000s)
ba2f8d4 lee@lee-GF75-Thin-9SC:~$ ~/Android/Sdk/platform-tools/adb -s emulator-5556 shell
2ce55ac generic_x86_64:/ $ su
000000e generic_x86_64:/ # cd data/local/tmp
000000e generic_x86_64:/data/local/tmp # ./user
0000000 lee@lee-GF75-Thin-9SC:~$
0000000000000 XMM15=00000000000000000000000000000000
                                    emu$ vi hw/android/goldfish/kvssd.c
```

17. 직접해보니까 이해가 잘 되는 것 같다.