Embedded System Practice
Program Assignment #1 Report

2016311821 한승하

Patch

이번 Programming assignment에서 kvssd.c의 기본 skeleton code는 iodev.c를 참고하였습니다.

수정이 필요하지 않은 부분은 단순히 iodev를 kvssd로 바꿔줌으로써 그대로 사용하였습니다.

Kvssd를 추가하기위해 다음과 같은 패치들을 진행해 주었습니다.

```
goldfish_segment_init();
goldfish_led_init();
goldfish_iodev_init();
goldfish_kvssd_init();
```

```
void goldfish_segment_init(void);
void goldfish_led_init(void);
void goldfish_iodev_init(void);
void goldfish_kvssd_init(void);
```

```
hw/android/goldfish/vmem.c \
    hw/android/goldfish/segments.c \
    hw/android/goldfish/leds.c \
    hw/android/goldfish/iodev.c \
    hw/android/goldfish/kvssd.c \
```

```
config GOLDFISH_KVSSD
        tristate "Android Goldfish kvssd"
        help
          kvssd driver for Android Goldfish emulator
```

```
obj-$(CONFIG_GOLDFISH_KVSSD) += goldfish_kvssd.o
```

Struct goldfish_kvssd_state

State는 다음과 같은 변수들로 구성하였습니다.

각각 key와 value를 받아 오기 위한 index들

kvssd.data파일과 kvssd.data파일의 file disciptor

Garbage collection 판단을 위한 free block, 등으로 구성 해 주었습니다.

```
struct goldfish_kvssd_state {
        struct goldfish_device dev;
        uint8_t exist;
        uint16_t key_index;
        uint16_t val_index;
        int kvssd_meta_fd;
        int kvssd_data_fd;
        uint32_t free_blk;
        uint32_t status;
        uint32_t cmd;
        uint32_t key[4];
        uint32_t value[1024];
};
```

Global variables

Kvssd에 필요한 global variables입니다.

Input 판단을 위한 define값들과 더불어

K2p table (meta data)를 위한 kvssd_meta_data

그리고 이 structur로 선언된 *k2p

새로 할당할 physical block numbe를 pointing하는 new_blk_ptr

DEV_WAIT상태에서의 Check를 위한 변수들과

Block들의 valid를 저장하고 있는 blk_valid로 구성되어있습니다.

```
#define N_BLK           20
#define N_KEY           20

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

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

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

```
struct kvssd_meta_data{
        uint32_t key[4];
};
```

```
struct kvssd_meta_data *k2p;

uint32_t new_blk_ptr=0;
int checking_process,check_key = 0;
char blk_valid[N_BLK];
```

goldfish_kvssd_init

```
void goldfish_kvssd_init(void)
{
        int idx,i;
        k2p = (struct kvssd_meta_data*)malloc(sizeof(struct kvssd_meta_data)* N_BLK);
        struct goldfish_kvssd_state *s;
        new_blk_ptr = 0;
        s = (struct goldfish_kvssd_state *)g_malloc0(sizeof(*s));
        s->dev.name = "goldfish_kvssd";
        s->dev.base = 0;
        s->dev.size = 0x1000;
        s->dev.irq_count = 1;
        s->dev.irq = 15;
        s->free_blk = N_BLK;
        s->kvssd_data_fd = open("/home/han/emu-2.2-release/external/qemu/kvssd.data", O_RDWR | O_CREAT, 0777);
        s->kvssd_meta_fd = open("/home/han/emu-2.2-release/external/qemu/kvssd.meta", O_RDWR | O_CREAT, 0777);
        goldfish_device_add(&s->dev, goldfish_kvssd_readfn, goldfish_kvssd_writefn, s);
        char *buf = (char*)malloc(sizeof(uint32_t));
        char *valid_check = (char*)malloc(sizeof(char)*N_BLK);
        int ret = read(s->kvssd_meta_fd,valid_check,sizeof(char)*N_BLK);
        if(ret == 0){
                for(idx=0;idx<N_BLK;idx++) write(s->kvssd_meta_fd,"0",1); //0 empty 1 vaild 2 invaild
                write(s->kvssd_meta_fd,"\n",1);
                for(idx=0;idx<N_BLK;idx++){
                        write(s->kvssd_meta_fd,"0000000000000000\n",(4*sizeof(uint32_t))+1);
                        for(i=0;i<4;i++) k2p[idx].key[i] = 0;
                        blk_valid[idx] = 'E';
                }

        }
```

<Meta file 구성>
(0,1,2 의 block valid 정보 N_BLK byte) ₩n
16Byte의 KEY ₩n
(N_BLK개)

Init에서는 기존의 iodev와 같은 방식으로 장치를 초기화하며, 이번 실습을 위해 iodev의 irq 15를 비활성화하고 kvssd에서 사용하였습니다. Kvssd파일들을 열어주고, 필요한 변수들은 선언하는 부분입니다.
이후 meta파일을 읽어 저장되어 있던 상태가 있는지 판단합니다.
읽은 값이 없다면 초기 설정으로 모두 0으로 pedding하고 모두 Empty로 처리해 줍니다.

goldfish_kvssd_init (con't)

```
else{
        for(idx=0;idx<N_BLK;idx++){
                switch(valid_check[idx]){
                        case '0':
                                blk_valid[idx] = 'E';
                                break;
                        case '1':
                                s->free_blk--;
                                blk_valid[idx] = 'V';
                                break;
                        case '2':
                                s->free_blk--;
                                blk_valid[idx] = 'I';
                                break;
                }
        }
        lseek(s->kvssd_meta_fd,1,SEEK_CUR);
        for(idx=0;idx<N_BLK;idx++){
                if(blk_valid[idx] == 'V'){
                        for(i=0;i<4;i++){
                                read(s->kvssd_meta_fd,buf,sizeof(uint32_t));
                                memcpy(&k2p[idx].key[i],buf,sizeof(uint32_t));
                        }
                }
        }
}
```

만약 읽은 값이 존재한다면, valid영역의 값으로 meta data를 구성해줍니다.
이후 key값을 차례로 읽어 k2p의 맞는 block에 저장해줍니다.

goldfish_kvssd_init (con't)

```
for(new_blk_ptr=0;new_blk_ptr<N_BLK;new_blk_ptr++) if(blk_valid[new_blk_ptr] == 'E') break;
for(idx=0;idx<N_BLK;idx++){
        if(blk_valid[idx] == 'V') lseek(s->kvssd_data_fd,sizeof(uint32_t)*1024+1,SEEK_CUR);
        else{
                for(i=0;i<sizeof(uint32_t)*1024;i++) write(s->kvssd_data_fd,"0000",sizeof(uint32_t));
                write(s->kvssd_data_fd,"\n",1);
        }
};

register_savevm(NULL,
                "goldfish_kvssd",
                0,
                KVSSD_STATE_SAVE_VERSION,
                goldfish_kvssd_save,
                goldfish_kvssd_load,
                s);
```

이후 Block를 초기화해줍니다. 읽어온 Valid 영역이 있다면 초기화 해주지 않습니다.

goldfish_kvssd_read

```
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->value[s->val_index];
                        s->val_index = (s->val_index+1)%1024;
                        return temp;
        };

        return 0;
}
```

서술 되어 있는 행동을 하도록 Read함수를 작성하였습니다.
각각 반환해야 할 값을 봔환해줍니다.

goldfish_kvssd_write (STATUS_REG)

```
static void goldfish_kvssd_write(void* opaque, hwaddr offset, uint32_t value)
{
        struct goldfish_kvssd_state* s = (struct goldfish_kvssd_state*)opaque;
        int idx,i;
        char *buf = (char*)malloc(sizeof(uint32_t));


        if ( offset < 0 ) {
                cpu_abort(cpu_single_env, "iodev_dev_read: Bad offset %" HWADDR_PRIx "\n", offset);
                return;
        }
        switch (offset) {
                case KVSSD_STATUS_REG:
                        if(s->status == DEV_WAIT){
                                if(checking_process == 3){
                                        if(check_key == 3){
                                                s->status = DEV_READY;
                                        }
                                        check_key = 0;
                                        checking_process = 0;
                                }
                                else{
                                        if(s->key[checking_process] == value) check_key++;
                                        checking_process++;
                                }
                        }
                        break;
```

Write함수입니다.
Status Reg는 WAIT상태 일 경우 쓰기 요청을 받습니다.
매번 들어온 값과 Key를 비교하여 4번의 요청 동안 같은 값이 확인 될 경우 DEV_READY로 Status를 바꾸어줍니다.

goldfish_kvssd_write (CMD_PUT)

```
case KVSSD_CMD_REG:
        switch(value){
                case CMD_PUT:
                        if(s->free_blk == 1) garbage_collection(s);
                        s->exist = 0;
                        for(idx=0;idx<N_BLK;idx++)
                        {
                                if(blk_valid[idx] == 'V' && memcmp(&s->key[0],&k2p[idx].key[0],sizeof(uint32_t)*4) == 0){
                                        blk_valid[idx] = 'I';
                                        lseek(s->kvssd_meta_fd,idx,SEEK_SET);
                                        write(s->kvssd_meta_fd,"2",1);
                                        s->exist = 1;
                                        break;
                                }
                        }
                        memcpy(&k2p[new_blk_ptr].key[0],&s->key[0],4*sizeof(uint32_t));
                        blk_valid[new_blk_ptr] = 'V';
                        lseek(s->kvssd_data_fd,(new_blk_ptr)*(1024*sizeof(uint32_t)+1),SEEK_SET);
                        for(idx=0;idx<1024;idx++){
                                memcpy(buf,&s->value[idx],sizeof(uint32_t));
                                write(s->kvssd_data_fd,buf,sizeof(uint32_t));
                        }
                        write(s->kvssd_data_fd,"\n",1);///data
                        lseek(s->kvssd_meta_fd,new_blk_ptr,SEEK_SET);
                        write(s->kvssd_meta_fd,"1",1);
                        lseek(s->kvssd_meta_fd,N_BLK+(new_blk_ptr*4*sizeof(uint32_t)+1),SEEK_SET);
                        write(s->kvssd_meta_fd,&s->key[0],sizeof(uint32_t)*4);
                        for(idx=0;idx<1024;idx++){
                                new_blk_ptr = (new_blk_ptr + 1)%1024;
                                if(blk_valid[new_blk_ptr] == 'E') break;
                        }
                        s->free_blk--;
                        s->status = DEV_READY;
                        break;
```

CMD PUT은 KEY값이 존재하는지 판단하고, 새로운 block을 할당하여 값을 저장하는 과정을 진행합니다. 이후 free blk를 줄여주고, new blk pointer를 변경해 줍니다.

goldfish_kvssd_write (CMD_GET)

```
case CMD_GET:
        s->exist = 0;
        for(idx=0;idx<N_BLK;idx++){
                if(blk_valid[idx] == 'V' && memcmp(&s->key[0],&k2p[idx].key[0],sizeof(uint32_t)*4) == 0){
                        s->exist = 1;
                        lseek(s->kvssd_data_fd,idx*(1024*sizeof(uint32_t)+1),SEEK_SET);
                        for(i=0;i<1024;i++){
                                read(s->kvssd_data_fd,buf,sizeof(uint32_t));
                                memcpy(&s->value[i],buf,sizeof(uint32_t));
                        }
                        break;
                }
        }
        s->status = DEV_WAIT;
        break;
```

Get은 해당 key가 있는지 k2p를 뒤져 확인합니다. Valid한 key값이 있다면 반환한 후 Wait으로 전환합니다.

goldfish_kvssd_write (CMD_ERASE)

```
case CMD_ERASE:
        s->exist = 0;
        for(idx=0;idx<N_BLK;idx++){
                if(blk_valid[idx] == 'V' && memcmp(&s->key[0],&k2p[idx].key[0],sizeof(uint32_t)*4) == 0){
                        s->exist = 1;
                        blk_valid[idx] = 'I';
                        lseek(s->kvssd_meta_fd,idx,SEEK_SET);
                        write(s->kvssd_meta_fd,"2",1);
                        break;
                }
        }
        s->status = DEV_READY;
        break;
```

Erase는 값을 찾아 invalid처리를 해주고 Ready상태로 전환합니다.

goldfish_kvssd_write (CMD_EXIST)

```
                break;

        case CMD_EXIST:
                s->exist = 0;
                for(idx=0;idx<N_BLK;idx++){
                        if(blk_valid[idx] == 'V' && memcmp(&s->key[0],&k2p[idx].key[0],sizeof(uint32_t)*4) == 0){
                                s->exist = 1;
                                break;
                        }
                }
                s->status = DEV_WAIT;
                break;
    }
    goldfish_device_set_irq(&s->dev, 0, 1);
    break;
```

Exist또한 찾은 값을 반환 후 Wait상태로 전환합니다.
이후 모든 cmd에 공통적인 irq를 처리해줍니다.

goldfish_kvssd_write (KEY_REG)

```
        case KVSSD_KEY_REG:
                s->key[s->key_index] = value;
                s->key_index = (s->key_index + 1)%4;
                break;
        case KVSSD_VALUE_REG:
                s->value[s->val_index] = value;
                s->val_index = (s->val_index + 1)%1024;
                break;
}
free(buf);
```

Key와 value reg에 대한 행동 또한 서술된 대로 처리해 주었습니다.

garbage_collection

```
void garbage_collection(struct goldfish_kvssd_state* s)
{
        int idx,i;
        for(idx=0;idx<N_BLK;idx++){
                if(blk_valid[idx] == 'I'){
                        lseek(s->kvssd_meta_fd,idx,SEEK_SET);
                        write(s->kvssd_meta_fd,"0",1);
                        lseek(s->kvssd_meta_fd,N_BLK+1+(idx*(sizeof(uint32_t)*4+1)),SEEK_SET);
                        write(s->kvssd_meta_fd,"0000000000000000\n",17);
                        lseek(s->kvssd_data_fd,idx*((sizeof(uint32_t)*1024)+1),SEEK_SET);
                        for(i=0;i<1024;i++) write(s->kvssd_data_fd,"0000",sizeof(uint32_t));
                        write(s->kvssd_data_fd,"\n",1);
                        blk_valid[idx] = 'E';
                        s->free_blk++;
                }
        }
}
```

Freeblock이 1개일 때 진행되는 GC는 모든 invalid block의 값을 지워 Empty block으로 바꾸어 줍니다.