Chapter 8

블록 디바이스 드라이버

15 February 2022

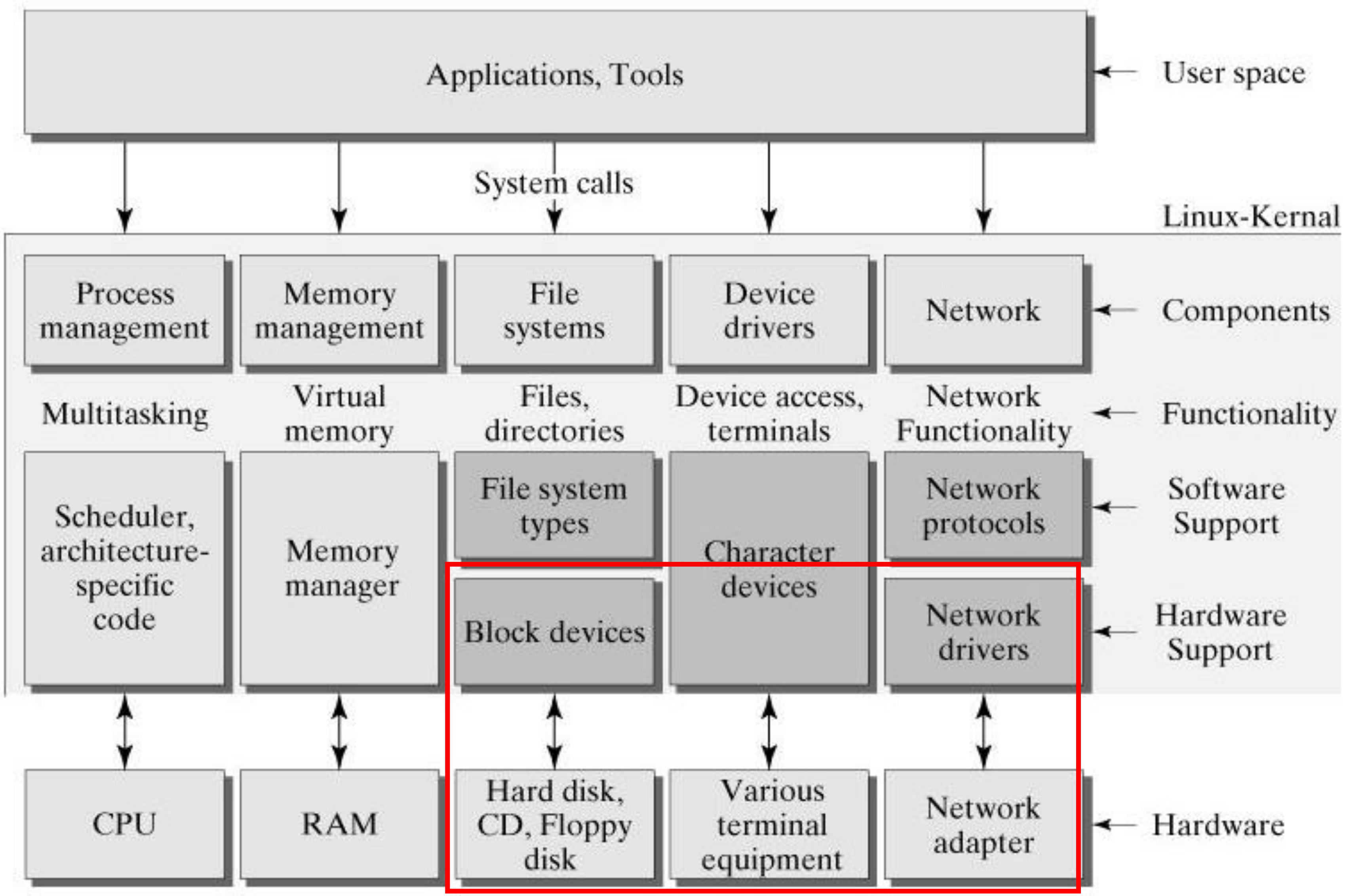
Minguk Choi
koreachoi96@gmail.com

Contents

- Block Device Driver
  - Block Device
  - 구조체
  - I/O scheduler

- Block Device Driver 실습

Linux Kernel Architecture



Block Device

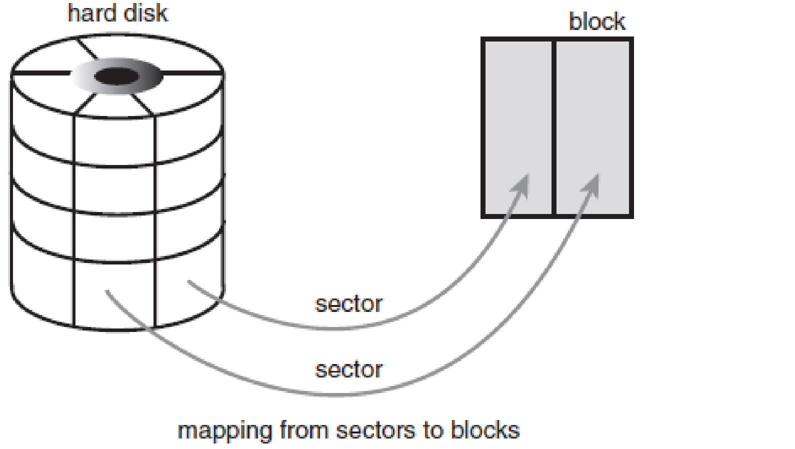
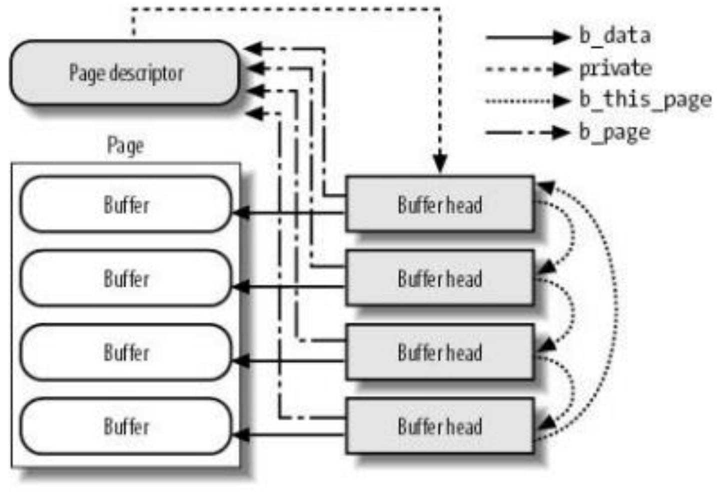


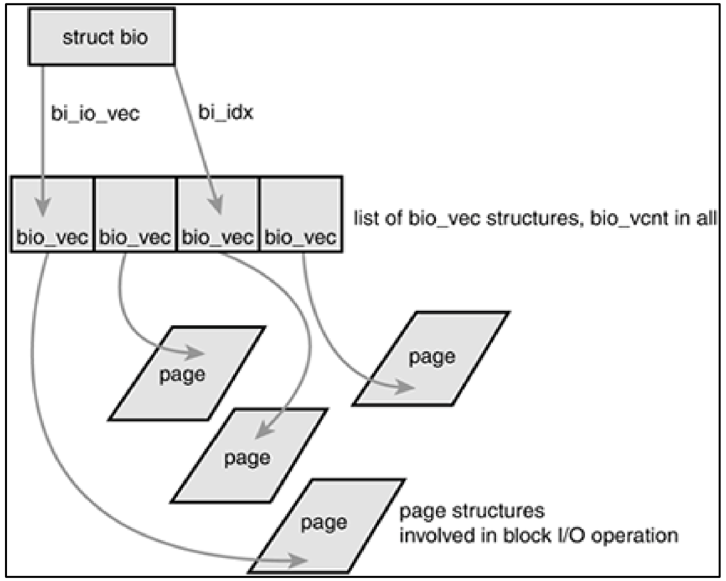
Figure 14.1 Relationship between sectors and blocks.

Figure 15-2. A buffer page including four buffers and their buffer heads



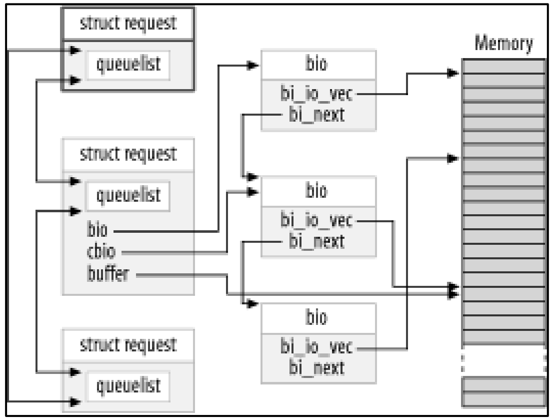
- 섹터(sector)
  - 블록 장치를 물리적으로 접근 가능한 최소 단위
  - 일반적으로 512B
- 블록(block)
  - 커널이 블록 장치를 논리적으로 접근 가능한 최소 단위
  - 일반적으로 512B, 1KB, 4KB 많이 사용
- 버퍼
  - 블록이 메모리 상에 존재할 때, 버퍼에 저장
  - 커널은 버퍼 관련 정보를 버퍼 헤드에 저장
  - 버퍼 헤드는 “버퍼-블록”의 연결관계를 나타냄

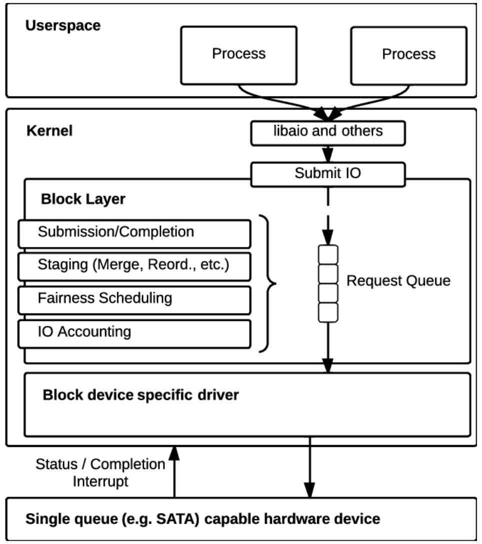
구조체



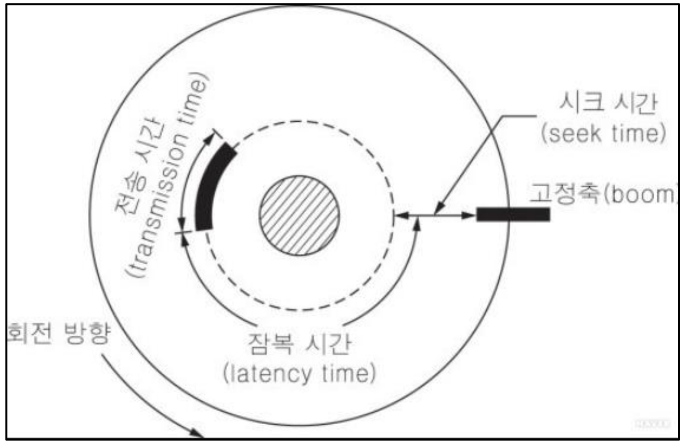
- struct bio
  - 커널 내부에서 블록 입출력을 전달하는 기본 장치
  - 현재 진행 중인 블록 입출력 동작을 세그먼트 리스트로 표현

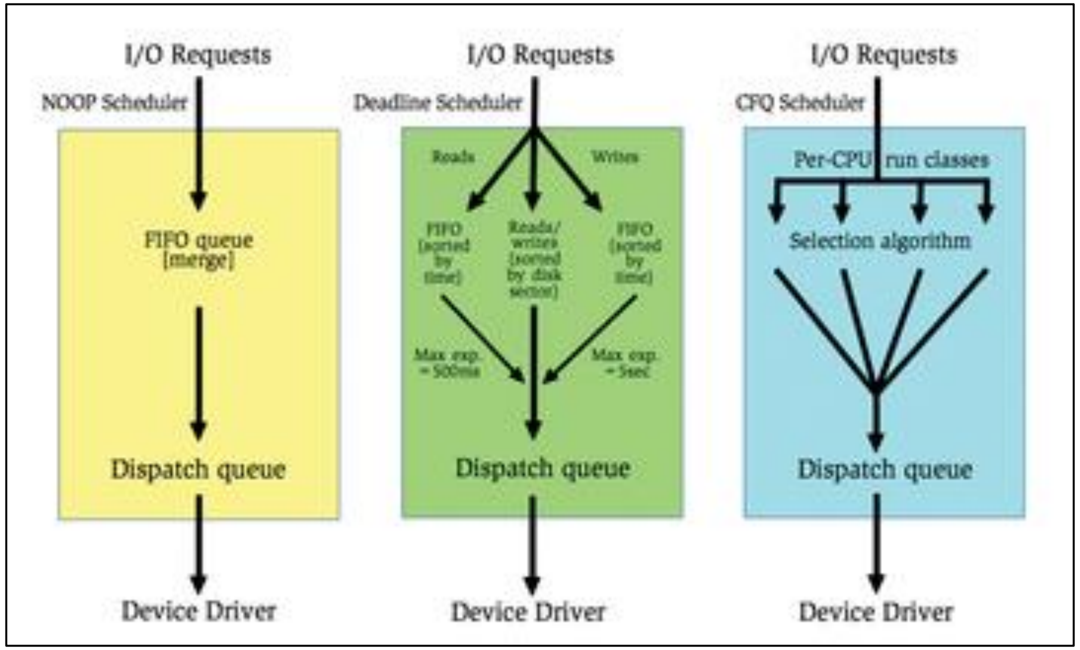
- struct bio_vec: 세그먼트
- struct bio
- struct request
- struct request_queue





I/O scheduler





- I/O scheduler
  - 커널이 입출력 요청 순서대로 블록에 전달하면, 성능이 나쁨
  - 현대 컴퓨터에서 가장 느린 동작 중 하나가 디스크 탐색
  - 하드 디스크 헤드를 특정 블록이 있는 헤드로 옮겨야 함

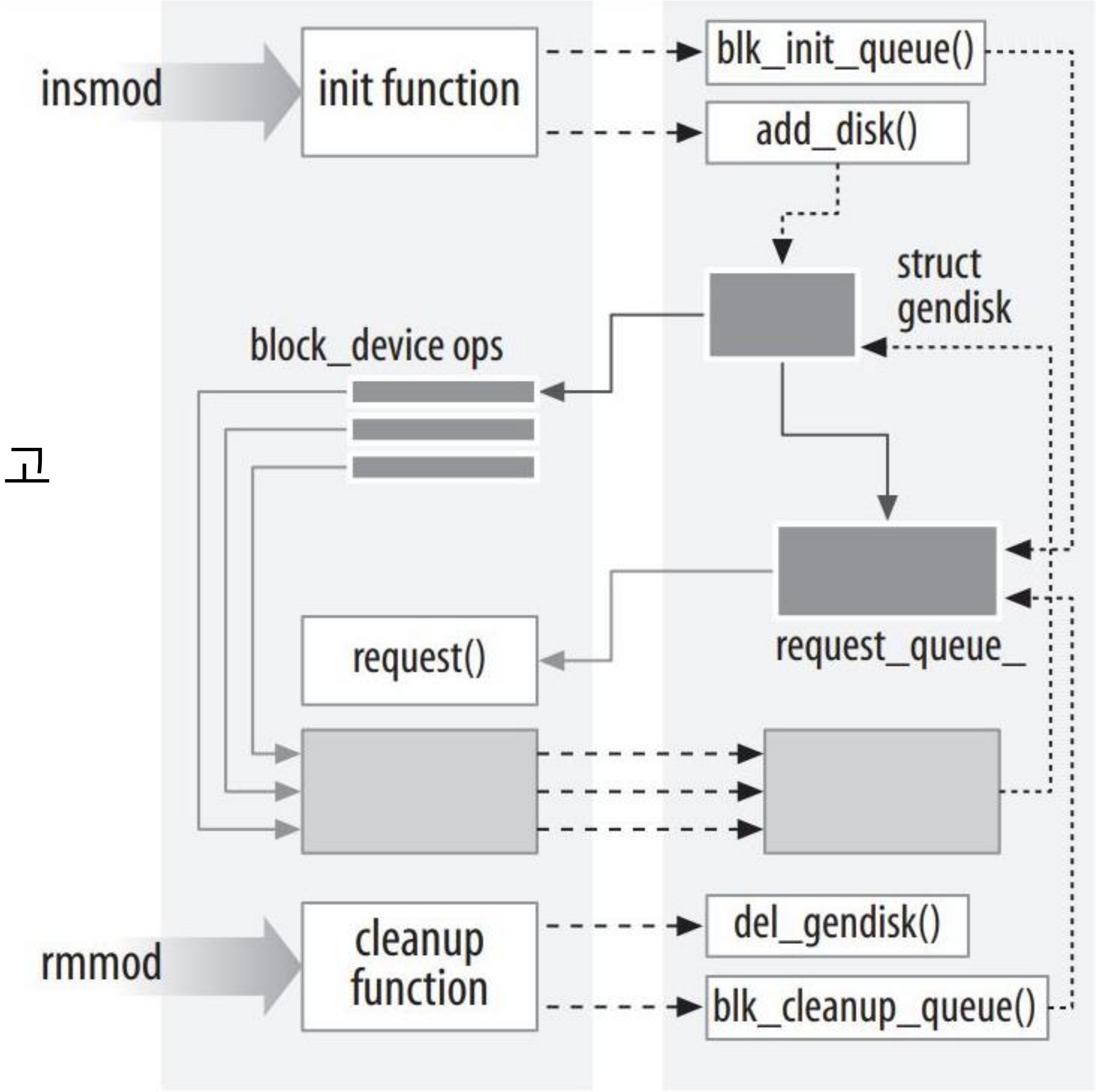
- 목적
  - 디스크 탐색 시간을 최소화 하여 보다 나은 전체 성능 향상

- 방법
  - 병합
    - ✓ 인접 섹터를 읽는 요청을 병합
  - 정렬
    - ✓ 섹터 순서에 따른 정렬 상태를 유치
    - ✓ 큐를 따라가며 탐색하는 동작이 디스크 섹터를 따라 순차적으로 일어나도록 함

- 종류
  - CFQ, Deadline, NOOP(SSD)

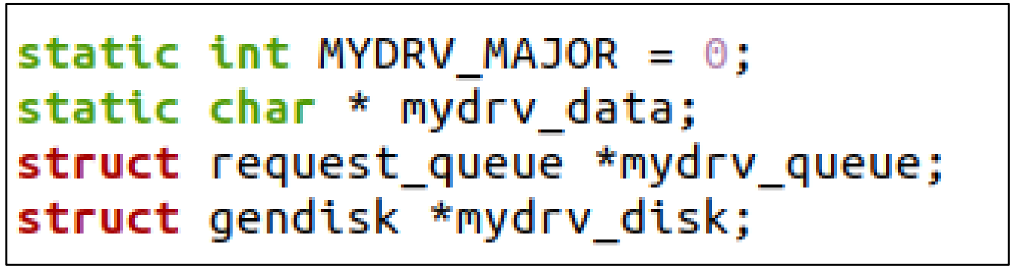
실습

- 리눅스 커널 내부구조
  - ch8 디바이스 드라이버
  - 블록 디바이스 드라이버 구현
- 구현 목표
  - 메모리의 일부를 가상 디스크 공간으로 할당하고
  - 해당 공간에 대한 I/O 제공
- 구현 내용
  - I/O request function
  - I/O operation function
    - ✓ open(), release(), ioctl()

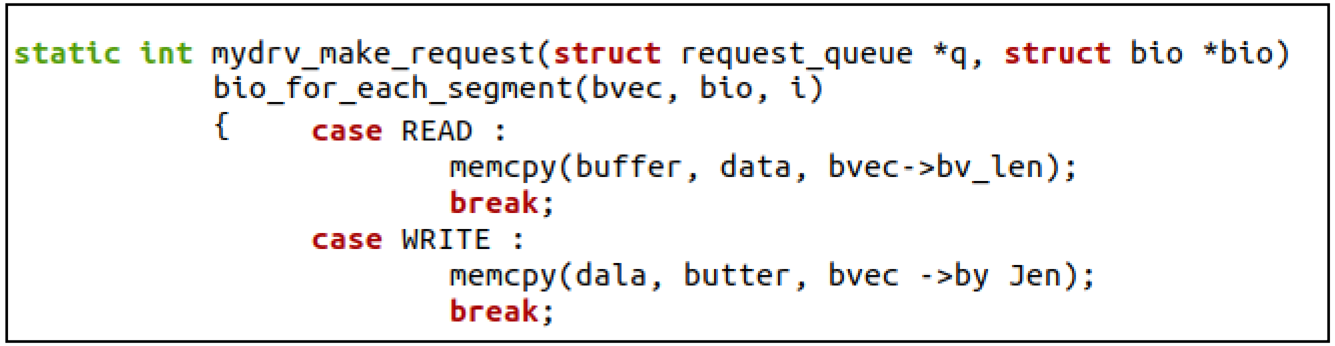


실습

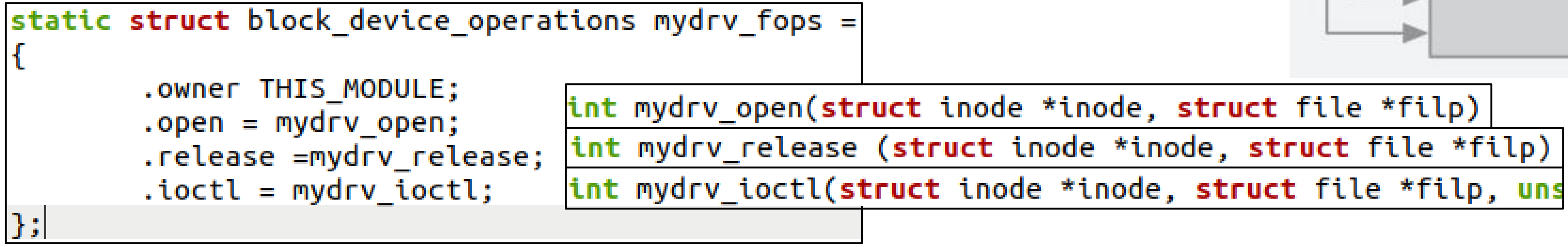
- I/O request/operation function



```
static int MYDRV_MAJOR = 0;
static char * mydrv_data;
struct request_queue *mydrv_queue;
struct gendisk *mydrv_disk;
```

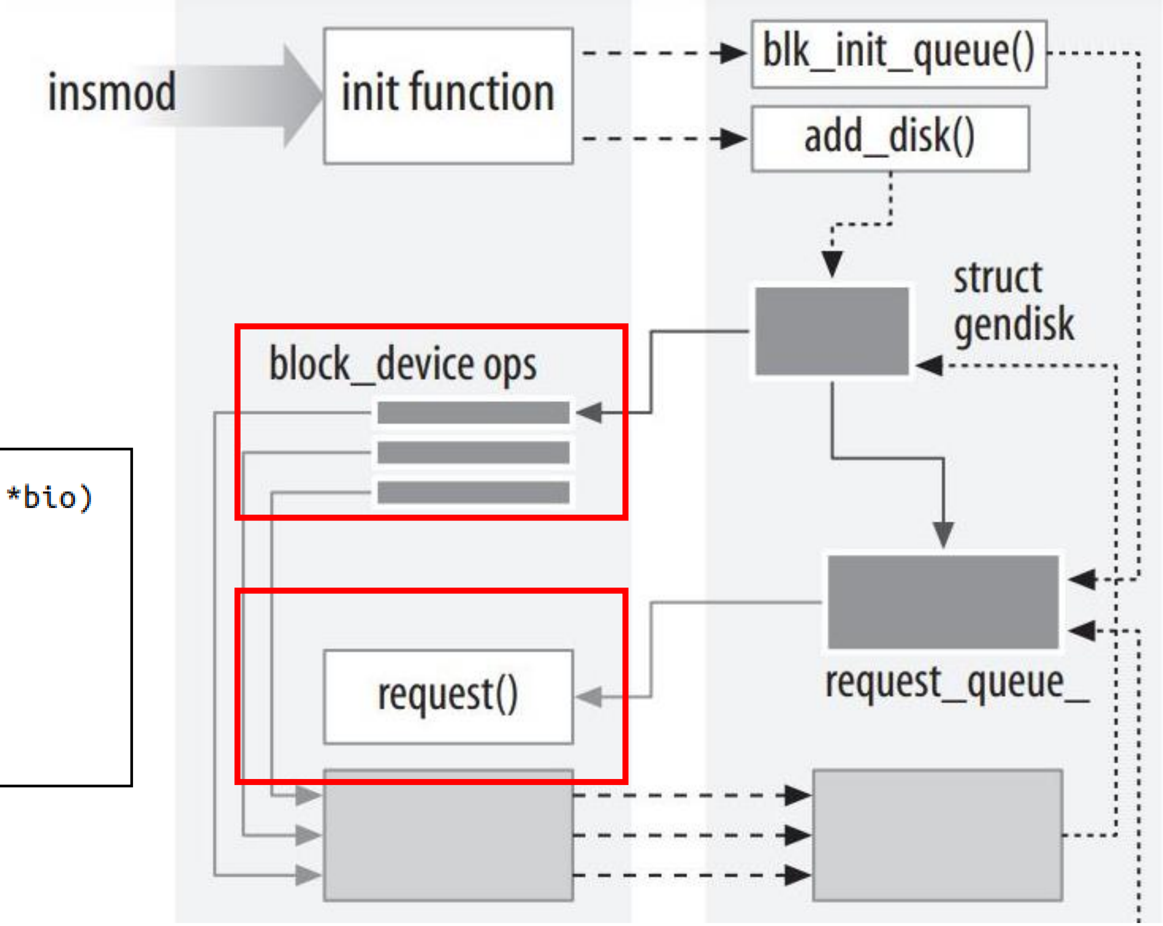


```
static int mydrv_make_request(struct request_queue *q, struct bio *bio)
           bio_for_each_segment(bvec, bio, i)
           {      case READ :
                         memcpy(buffer, data, bvec->bv_len);
                         break;
                  case WRITE :
                         memcpy(dala, butter, bvec ->by Jen);
                         break;
```



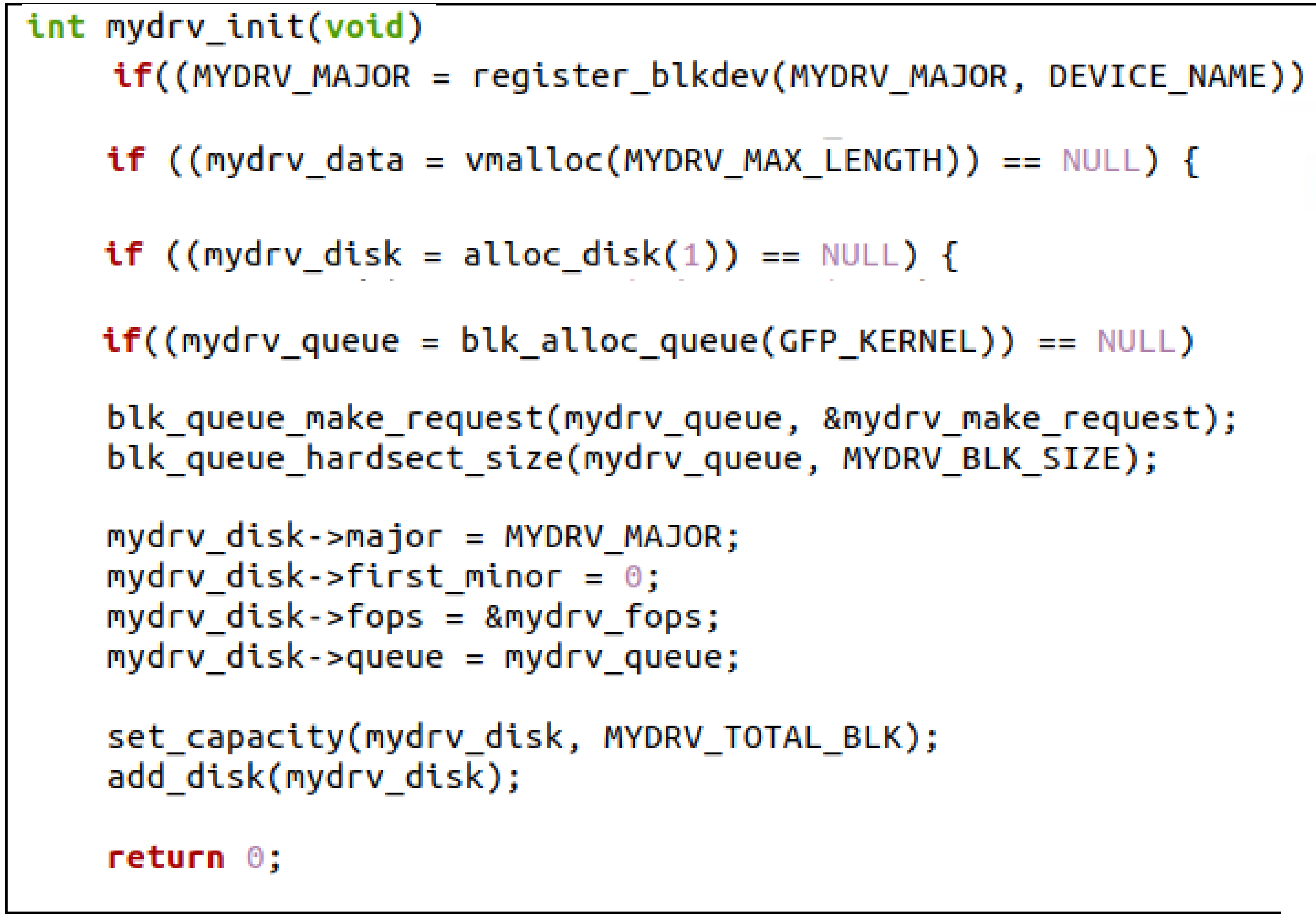
```
static struct block_device_operations mydrv_fops =
{
        .owner THIS_MODULE;
        .open = mydrv_open;
        .release =mydrv_release;
        .ioctl = mydrv_ioctl;
};
```

```
int mydrv_open(struct inode *inode, struct file *filp)
int mydrv_release (struct inode *inode, struct file *filp)
int mydrv_ioctl(struct inode *inode, struct file *filp, uns
```

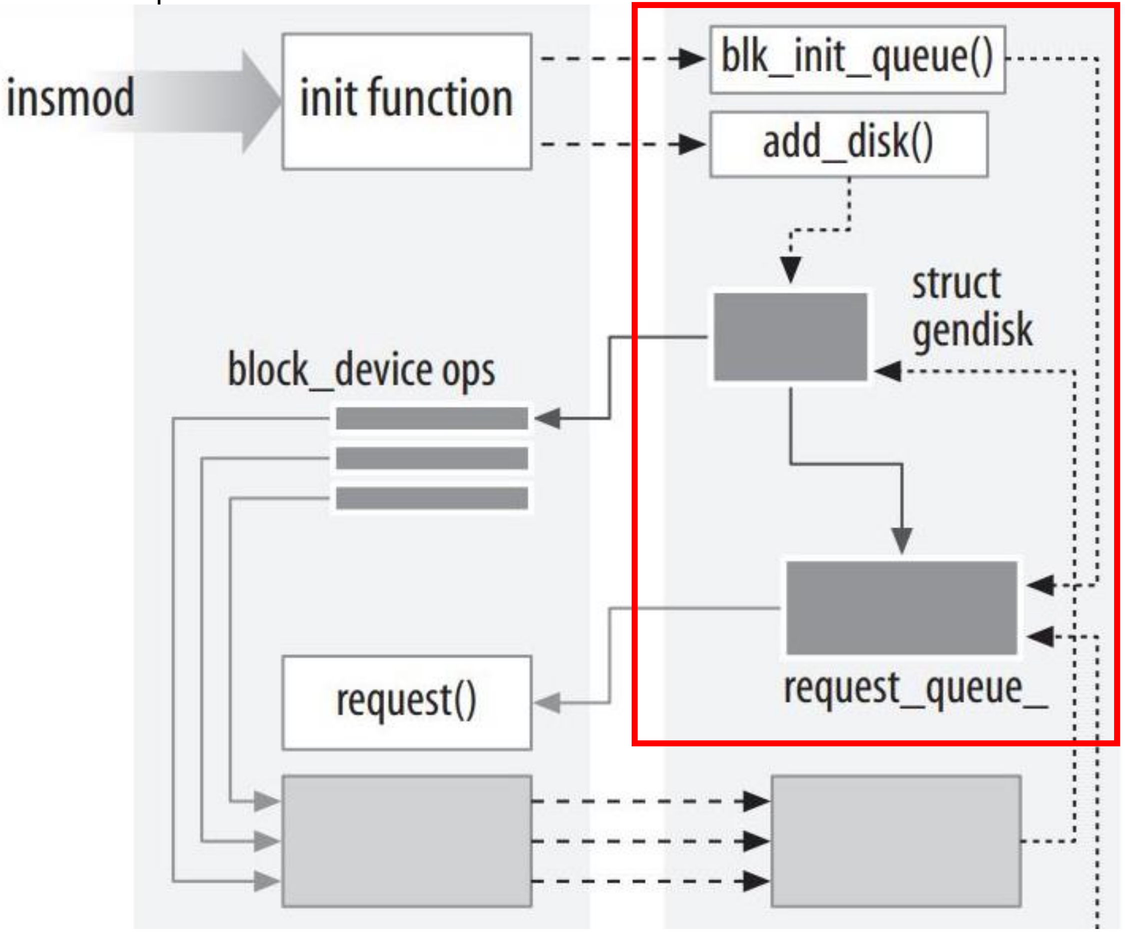


실습

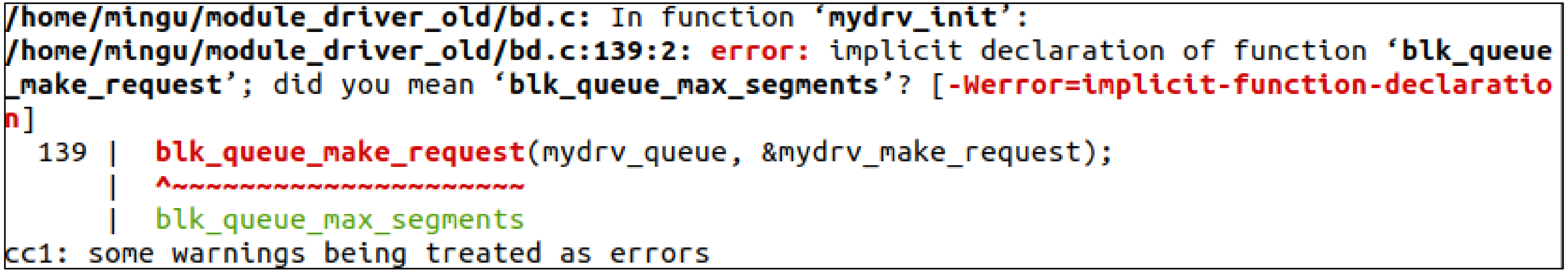
- Init block device driver



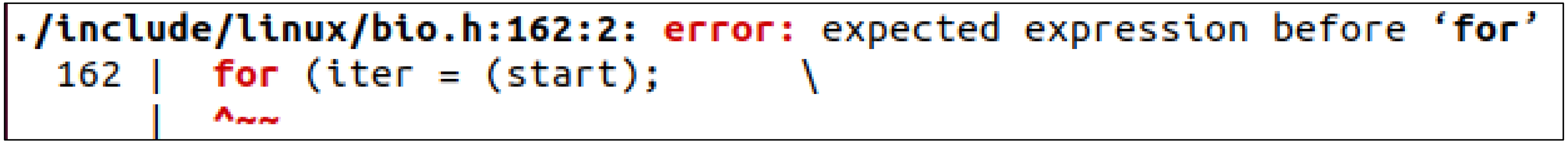
```
int mydrv_init(void)
    if((MYDRV_MAJOR = register_blkdev(MYDRV_MAJOR, DEVICE_NAME)) < 0) {

    if ((mydrv_data = vmalloc(MYDRV_MAX_LENGTH)) == NULL) {

    if ((mydrv_disk = alloc_disk(1)) == NULL) {

    if((mydrv_queue = blk_alloc_queue(GFP_KERNEL)) == NULL)

    blk_queue_make_request(mydrv_queue, &mydrv_make_request);
    blk_queue_hardsect_size(mydrv_queue, MYDRV_BLK_SIZE);

    mydrv_disk->major = MYDRV_MAJOR;
    mydrv_disk->first_minor = 0;
    mydrv_disk->fops = &mydrv_fops;
    mydrv_disk->queue = mydrv_queue;

    set_capacity(mydrv_disk, MYDRV_TOTAL_BLK);
    add_disk(mydrv_disk);

    return 0;
```



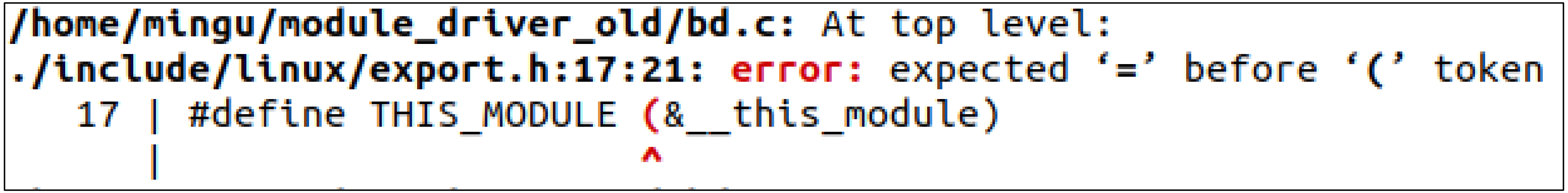
컴파일 에러



```
/home/mingu/module_driver_old/bd.c: In function ‘mydrv_init’:
/home/mingu/module_driver_old/bd.c:139:2: error: implicit declaration of function ‘blk_queue
_make_request’; did you mean ‘blk_queue_max_segments’? [-Werror=implicit-function-declaratio
n]
  139 |  blk_queue_make_request(mydrv_queue, &mydrv_make_request);
      |  ^~~~~~~~~~~~~~~~~~~~~~
      |  blk_queue_max_segments
cc1: some warnings being treated as errors
```



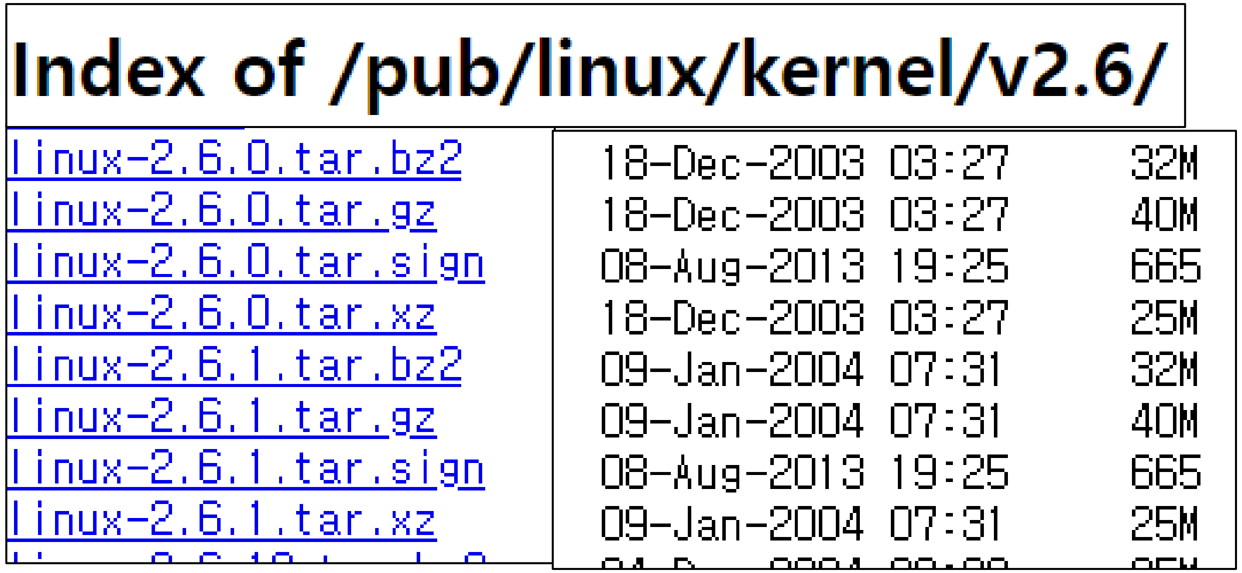
```
./include/linux/bio.h:162:2: error: expected expression before ‘for’
  162 |  for (iter = (start);     \
      |  ^~~
```



```
/home/mingu/module_driver_old/bd.c: At top level:
./include/linux/export.h:17:21: error: expected ‘=’ before ‘(’ token
   17 | #define THIS_MODULE (&__this_module)
      |                     ^
```

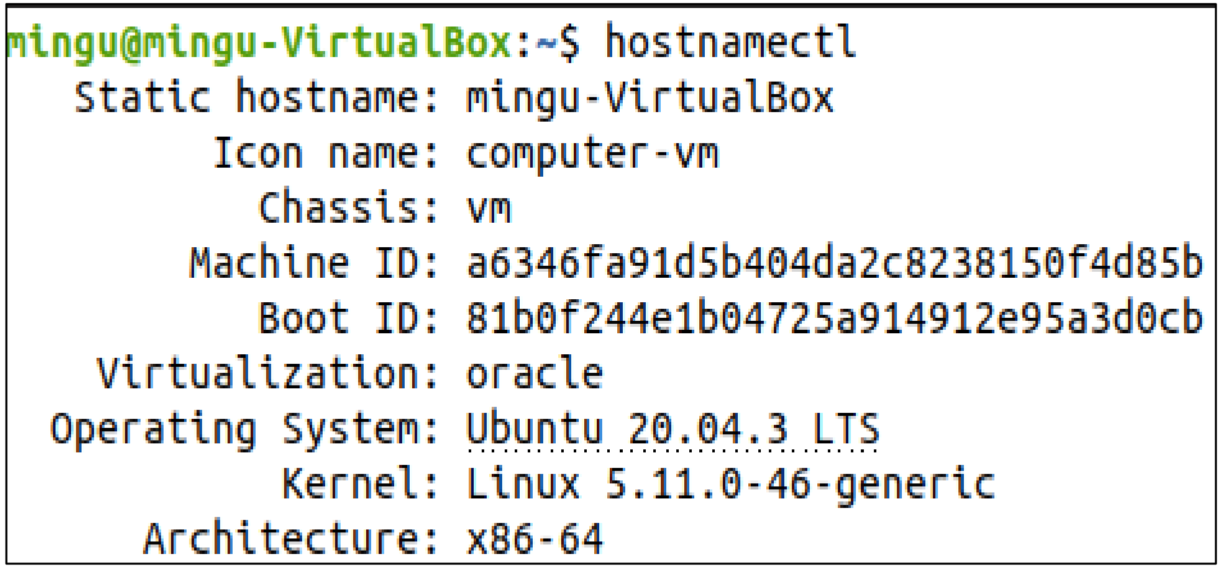
컴파일 에러 원인

- 리눅스 커널 내부구조
  - Kernel: Linux 2.6 (2003년)
- VirtualBox
  - Kernel: Linux 5.11.0
- $ hostnamectl



Index of /pub/linux/kernel/v2.6/

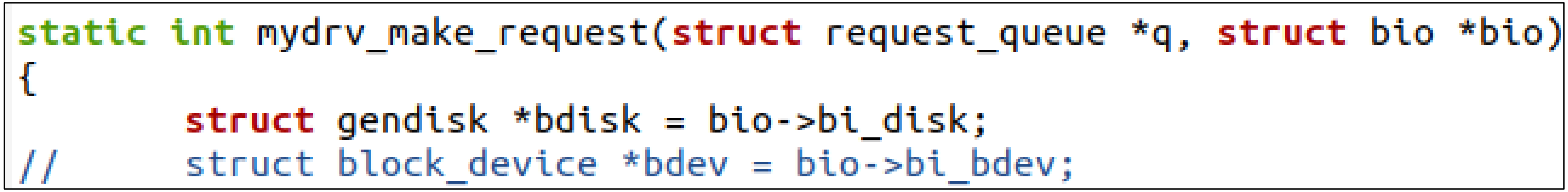
```
linux-2.6.0.tar.bz2      18-Dec-2003 03:27    32M
linux-2.6.0.tar.gz       18-Dec-2003 03:27    40M
linux-2.6.0.tar.sign     08-Aug-2013 19:25    665
linux-2.6.0.tar.xz       18-Dec-2003 03:27    25M
linux-2.6.1.tar.bz2      09-Jan-2004 07:31    32M
linux-2.6.1.tar.gz       09-Jan-2004 07:31    40M
linux-2.6.1.tar.sign     08-Aug-2013 19:25    665
linux-2.6.1.tar.xz       09-Jan-2004 07:31    25M
```



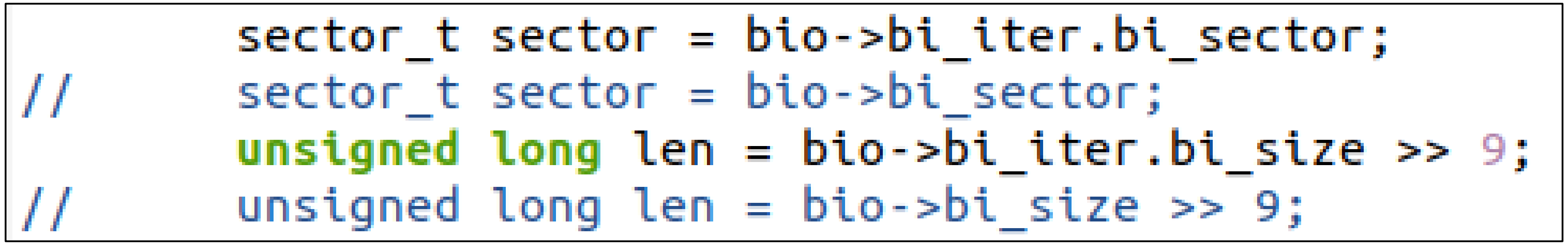
```
mingu@mingu-VirtualBox:~$ hostnamectl
   Static hostname: mingu-VirtualBox
         Icon name: computer-vm
           Chassis: vm
        Machine ID: a6346fa91d5b404da2c8238150f4d85b
           Boot ID: 81b0f244e1b04725a914912e95a3d0cb
    Virtualization: oracle
  Operating System: Ubuntu 20.04.3 LTS
            Kernel: Linux 5.11.0-46-generic
      Architecture: x86-64
```

실습 목표

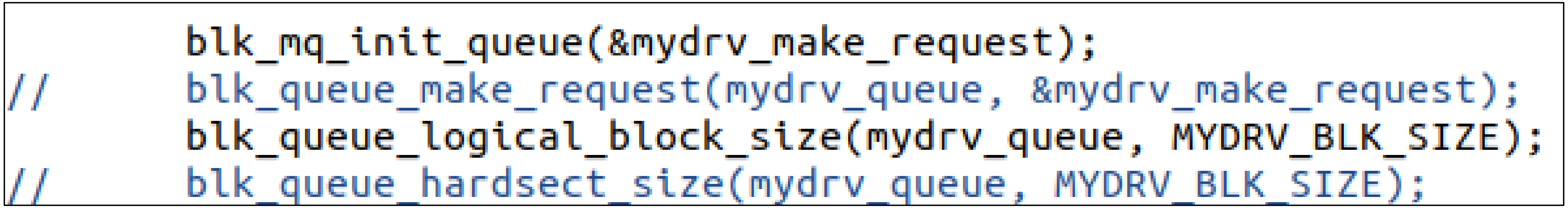
- 커널 함수를 업데이트 하자!



```
static int mydrv_make_request(struct request_queue *q, struct bio *bio)
{
        struct gendisk *bdisk = bio->bi_disk;
//      struct block_device *bdev = bio->bi_bdev;
```

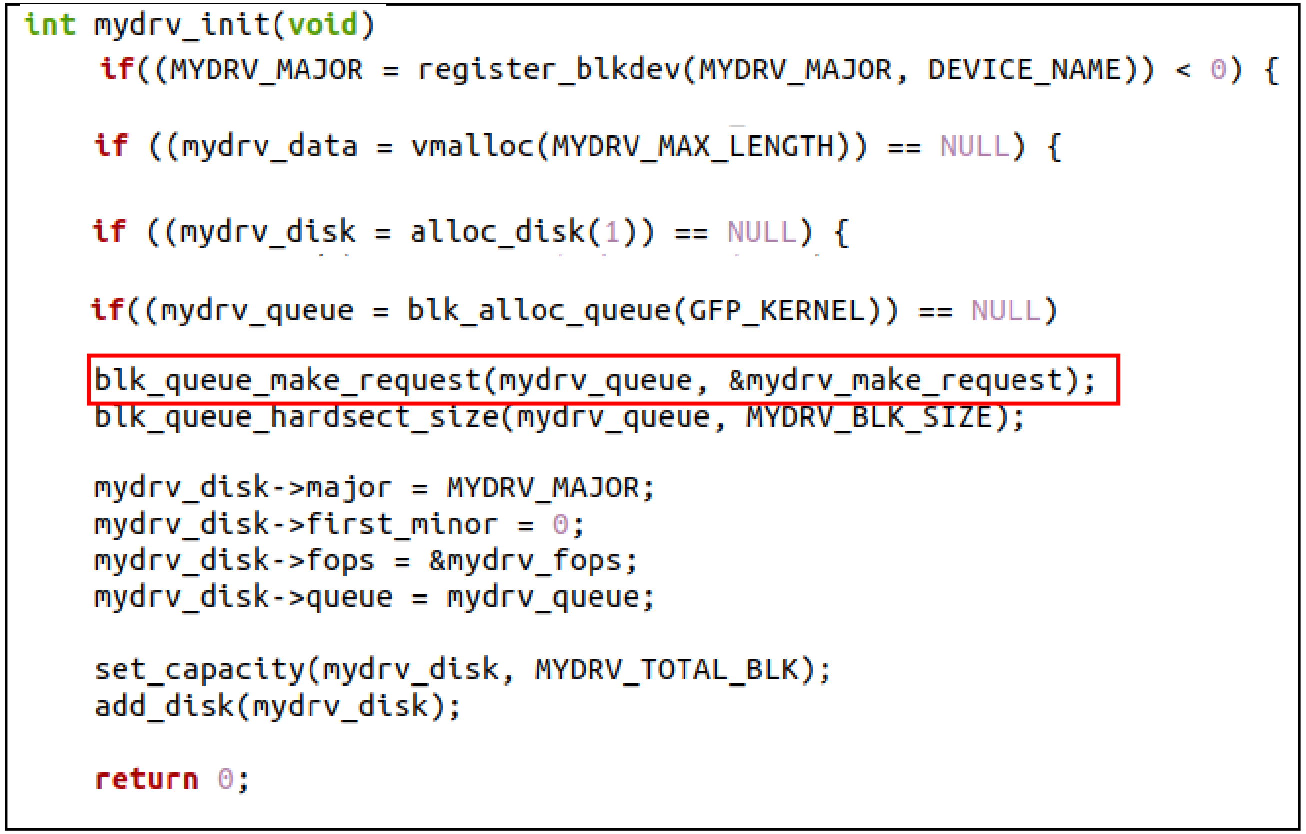


```
        sector_t sector = bio->bi_iter.bi_sector;
//      sector_t sector = bio->bi_sector;
        unsigned long len = bio->bi_iter.bi_size >> 9;
//      unsigned long len = bio->bi_size >> 9;
```



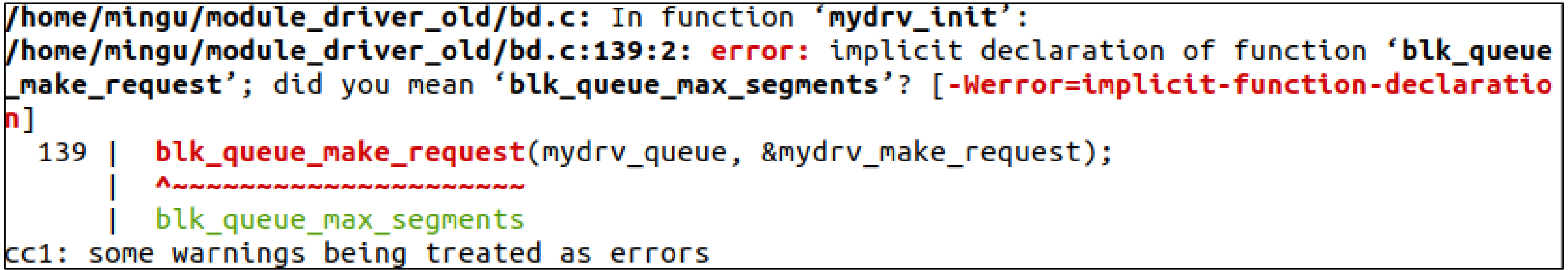
```
        blk_mq_init_queue(&mydrv_make_request);
//      blk_queue_make_request(mydrv_queue, &mydrv_make_request);
        blk_queue_logical_block_size(mydrv_queue, MYDRV_BLK_SIZE);
//      blk_queue_hardsect_size(mydrv_queue, MYDRV_BLK_SIZE);
```

blk_queue_make_request()



```
int mydrv_init(void)
     if((MYDRV_MAJOR = register_blkdev(MYDRV_MAJOR, DEVICE_NAME)) < 0) {

    if ((mydrv_data = vmalloc(MYDRV_MAX_LENGTH)) == NULL) {

    if ((mydrv_disk = alloc_disk(1)) == NULL) {

    if((mydrv_queue = blk_alloc_queue(GFP_KERNEL)) == NULL)

    blk_queue_make_request(mydrv_queue, &mydrv_make_request);
    blk_queue_hardsect_size(mydrv_queue, MYDRV_BLK_SIZE);

    mydrv_disk->major = MYDRV_MAJOR;
    mydrv_disk->first_minor = 0;
    mydrv_disk->fops = &mydrv_fops;
    mydrv_disk->queue = mydrv_queue;

    set_capacity(mydrv_disk, MYDRV_TOTAL_BLK);
    add_disk(mydrv_disk);

    return 0;
```

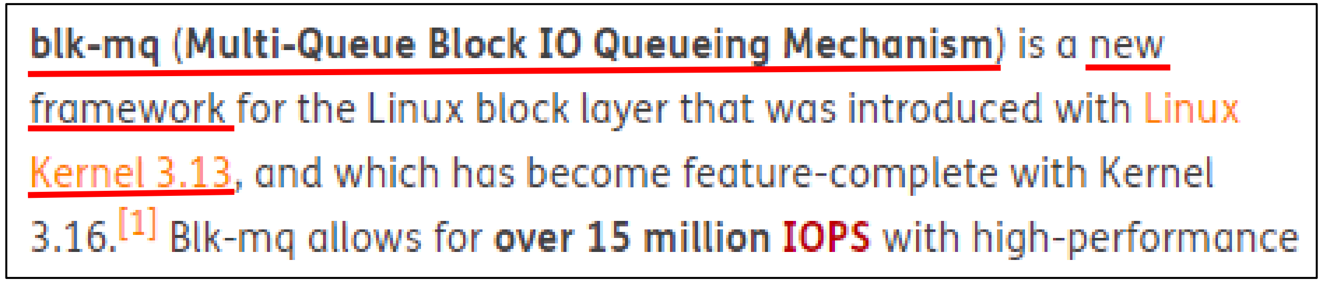
- blk_queue_make_request()
  - define an alternate make_request function for a device



```
/home/mingu/module_driver_old/bd.c: In function ‘mydrv_init’:
/home/mingu/module_driver_old/bd.c:139:2: error: implicit declaration of function ‘blk_queue
_make_request’; did you mean ‘blk_queue_max_segments’? [-Werror=implicit-function-declaratio
n]
  139 |   blk_queue_make_request(mydrv_queue, &mydrv_make_request);
      |   ^~~~~~~~~~~~~~~~~~~~~~
      |   blk_queue_max_segments
cc1: some warnings being treated as errors
```

Multi-Queue Block I/O (blk-mq)

- Multi-Queue Block I/O
  - why? single-queue is bottleneck
  - Multi-processor, SSD



blk-mq (**Multi-Queue Block IO Queueing Mechanism**) is a new framework for the Linux block layer that was introduced with Linux Kernel 3.13, and which has become feature-complete with Kernel 3.16.[1] Blk-mq allows for **over 15 million IOPS** with high-performance

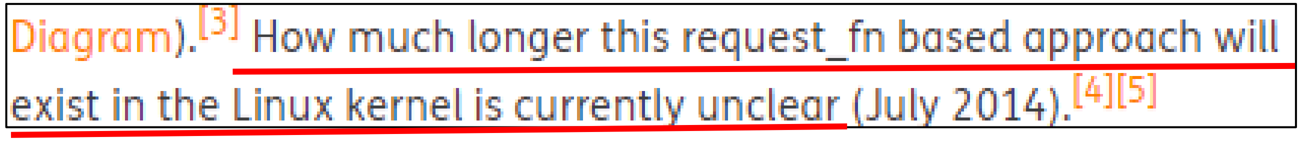
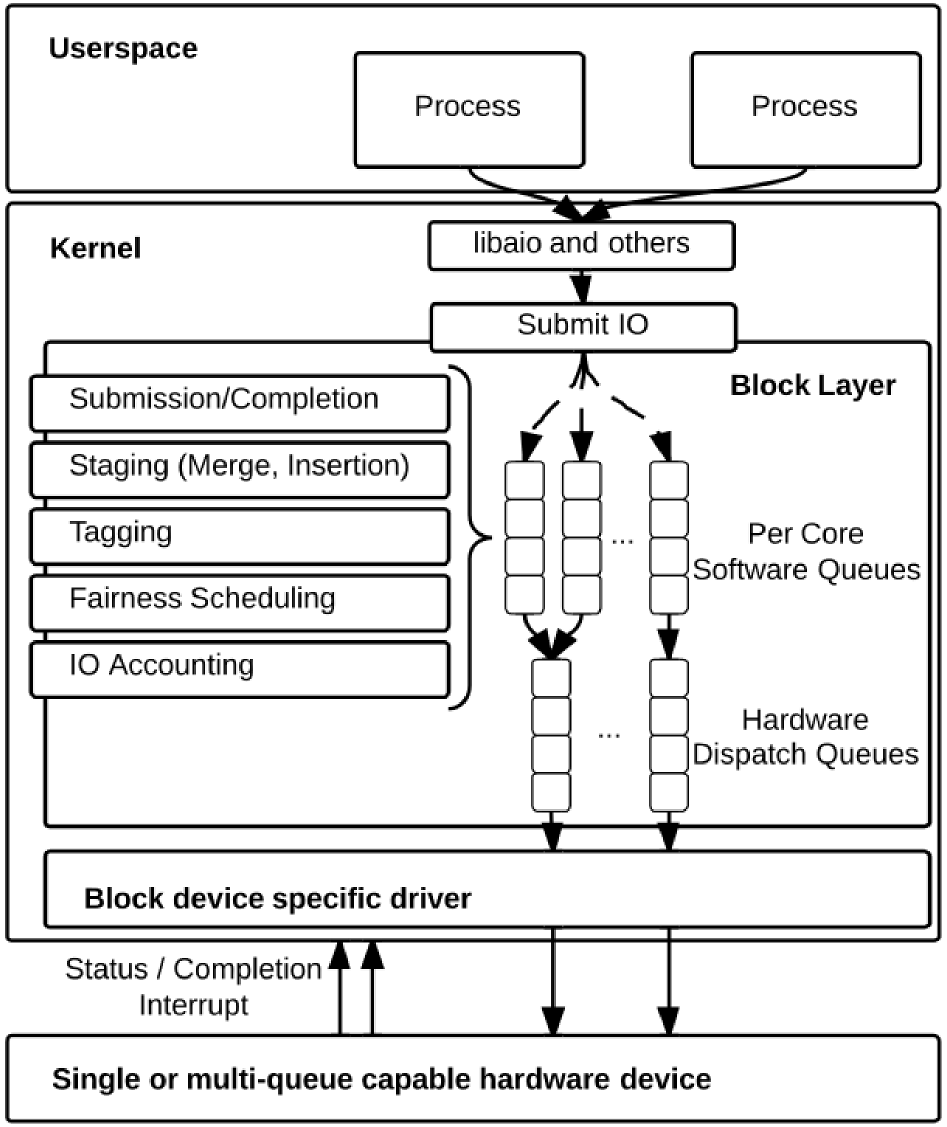


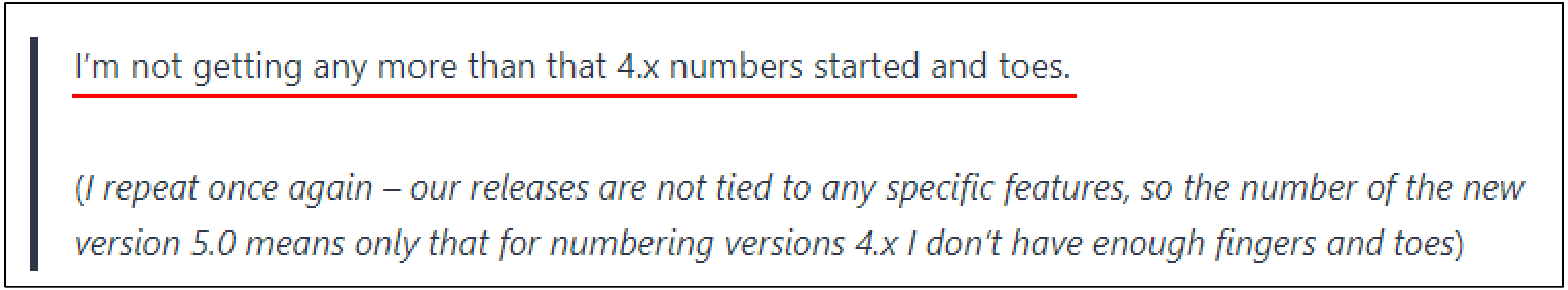
Diagram).[3] How much longer this request_fn based approach will exist in the Linux kernel is currently unclear (July 2014).[4][5]



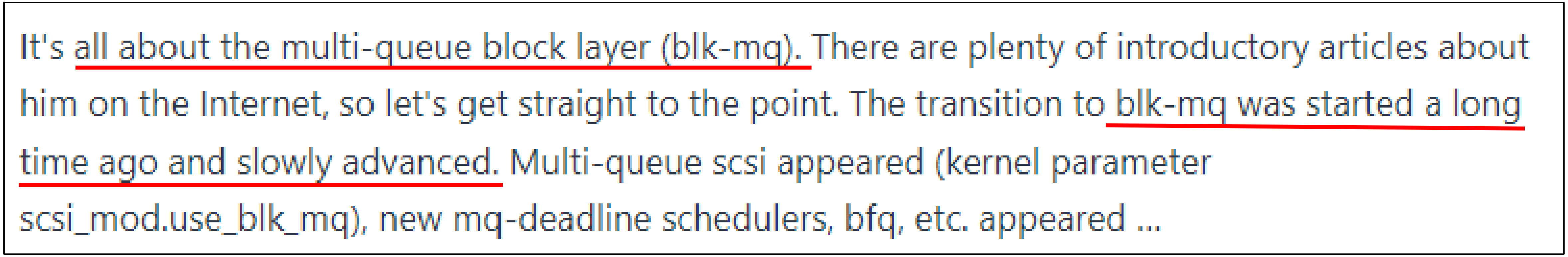
Linux Kernel 5.0

Updated 10 March 2019

Linus Torvalds
torvalds



> I'm not getting any more than that 4.x numbers started and toes.
>
> *(I repeat once again – our releases are not tied to any specific features, so the number of the new version 5.0 means only that for numbering versions 4.x I don't have enough fingers and toes)*

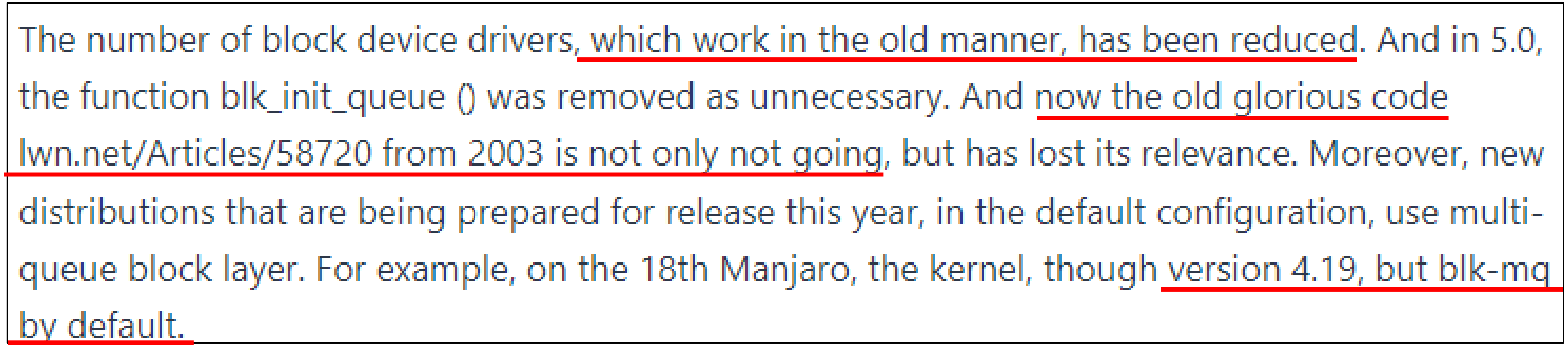


It's all about the multi-queue block layer (blk-mq). There are plenty of introductory articles about him on the Internet, so let's get straight to the point. The transition to blk-mq was started a long time ago and slowly advanced. Multi-queue scsi appeared (kernel parameter scsi_mod.use_blk_mq), new mq-deadline schedulers, bfq, etc. appeared ...

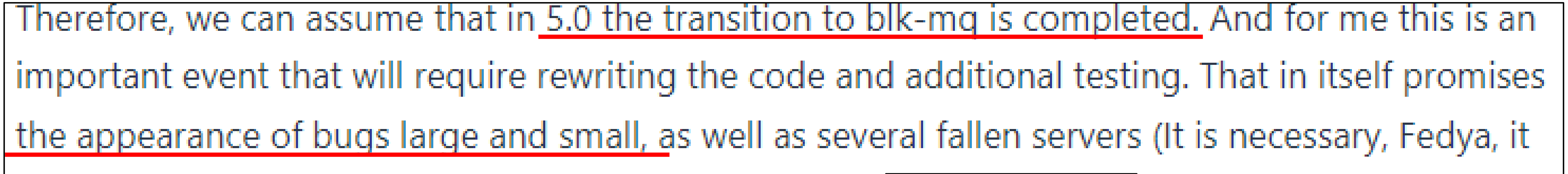
- https://prog.world/linux-kernel-5-0-we-write-simple-block-device-under-blk-mq/

Linux Kernel 5.0

Updated 10 March 2019

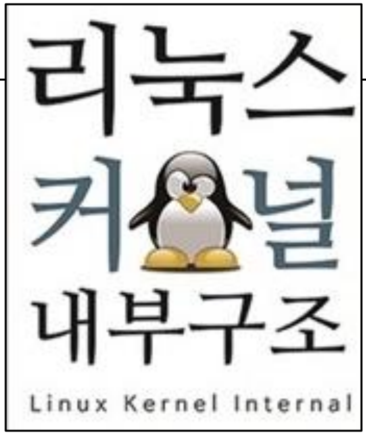


The number of block device drivers, which work in the old manner, has been reduced. And in 5.0, the function blk_init_queue () was removed as unnecessary. And now the old glorious code lwn.net/Articles/58720 from 2003 is not only not going, but has lost its relevance. Moreover, new distributions that are being prepared for release this year, in the default configuration, use multi-queue block layer. For example, on the 18th Manjaro, the kernel, though version 4.19, but blk-mq by default.



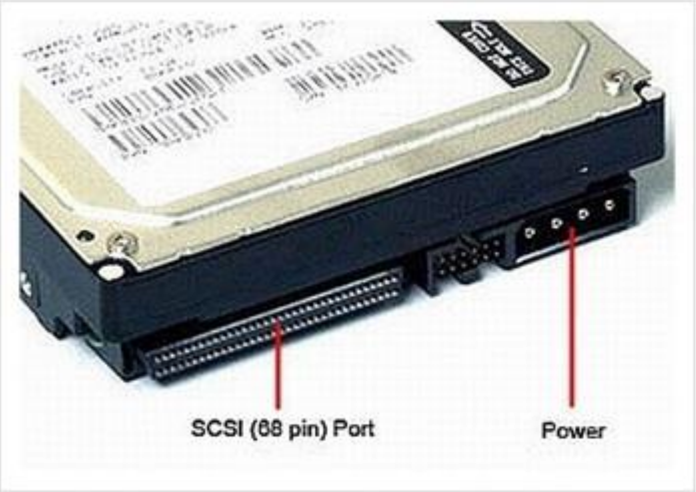
Therefore, we can assume that in 5.0 the transition to blk-mq is completed. And for me this is an important event that will require rewriting the code and additional testing. That in itself promises the appearance of bugs large and small, as well as several fallen servers (It is necessary, Fedya, it is necessary! (C)).

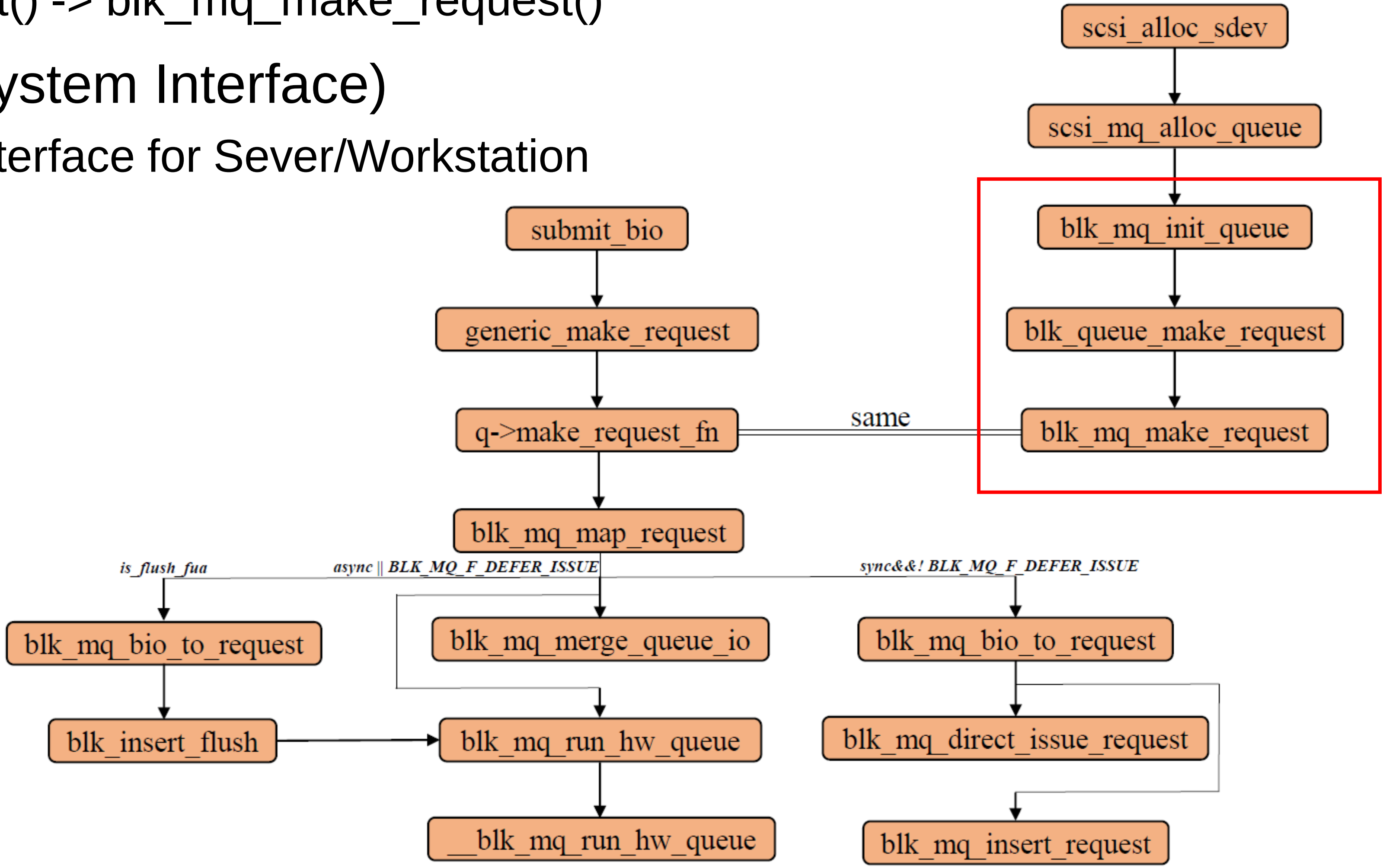
https://prog.world/linux-kernel-5-0-we-write-simple-block-device-under-blk-mq/



MQ processing structure flow diagram

- Kernel 3.xx
  - blk_queue_make_request() -> blk_mq_make_request()
- SCSI(Small Computer System Interface)
  - Fast but Expensive I/O Interface for Sever/Workstation





scsi_alloc_sdev

scsi_mq_alloc_queue

blk_mq_init_queue

blk_queue_make_request

blk_mq_make_request

submit_bio

generic_make_request

q->make_request_fn

same

blk_mq_map_request

is_flush_fua

async || BLK_MQ_F_DEFER_ISSUE

sync&&! BLK_MQ_F_DEFER_ISSUE

blk_mq_bio_to_request

blk_insert_flush

blk_mq_merge_queue_io

blk_mq_run_hw_queue

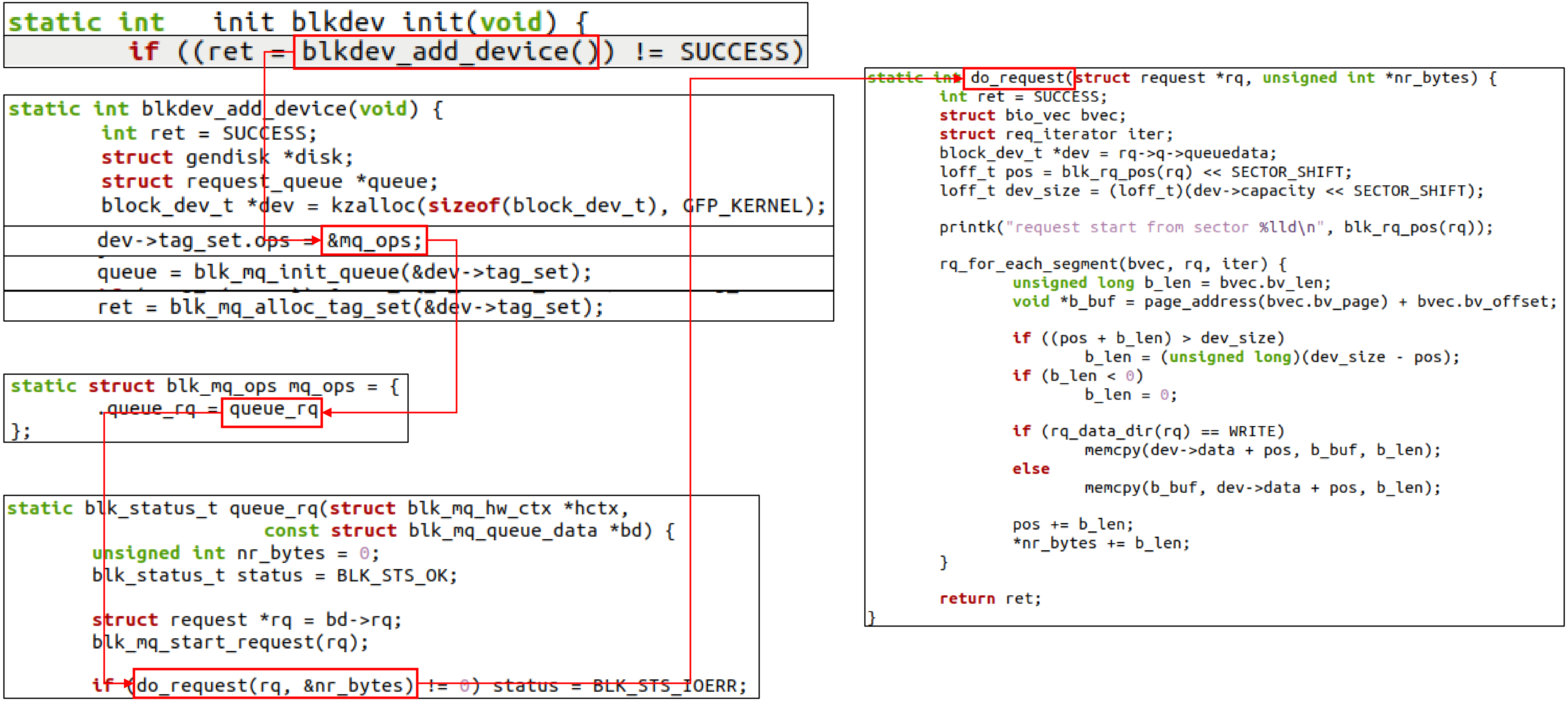
__blk_mq_run_hw_queue

blk_mq_bio_to_request

blk_mq_direct_issue_request

blk_mq_insert_request

What’s different?



```
static int __init blkdev_init(void) {
        if ((ret = blkdev_add_device()) != SUCCESS)
```

```
static int blkdev_add_device(void) {
        int ret = SUCCESS;
        struct gendisk *disk;
        struct request_queue *queue;
        block_dev_t *dev = kzalloc(sizeof(block_dev_t), GFP_KERNEL);
        dev->tag_set.ops = &mq_ops;
        queue = blk_mq_init_queue(&dev->tag_set);
        ret = blk_mq_alloc_tag_set(&dev->tag_set);
```

```
static struct blk_mq_ops mq_ops = {
        .queue_rq = queue_rq
};
```

```
static blk_status_t queue_rq(struct blk_mq_hw_ctx *hctx,
                             const struct blk_mq_queue_data *bd) {
        unsigned int nr_bytes = 0;
        blk_status_t status = BLK_STS_OK;

        struct request *rq = bd->rq;
        blk_mq_start_request(rq);

        if (do_request(rq, &nr_bytes) != 0) status = BLK_STS_IOERR;
```

```
static int do_request(struct request *rq, unsigned int *nr_bytes) {
        int ret = SUCCESS;
        struct bio_vec bvec;
        struct req_iterator iter;
        block_dev_t *dev = rq->q->queuedata;
        loff_t pos = blk_rq_pos(rq) << SECTOR_SHIFT;
        loff_t dev_size = (loff_t)(dev->capacity << SECTOR_SHIFT);

        printk("request start from sector %lld\n", blk_rq_pos(rq));

        rq_for_each_segment(bvec, rq, iter) {
                unsigned long b_len = bvec.bv_len;
                void *b_buf = page_address(bvec.bv_page) + bvec.bv_offset;

                if ((pos + b_len) > dev_size)
                        b_len = (unsigned long)(dev_size - pos);
                if (b_len < 0)
                        b_len = 0;

                if (rq_data_dir(rq) == WRITE)
                        memcpy(dev->data + pos, b_buf, b_len);
                else
                        memcpy(b_buf, dev->data + pos, b_len);

                pos += b_len;
                *nr_bytes += b_len;
        }

        return ret;
}
```

Q & A

- Block Device Driver
- Block Device Driver 구현 실습
  - 컴파일 에러
- Multi-Queue Block I/O
- Linux kernel 5
- Block Device Driver 구현 실습
  - Linux kernel 5