Project 3: The Kernel Crypto API

Group 10-03: Behnam Saeedi, zhaoheng wang, Levi Willmeth
CS444: Operating systems II

Spring 2017

Abstract

CONTENTS

1 DESIGN

1.1 Description of concept

Write a disk device driver module for linux-yocto-3.14 that can perform encryption based on a key passed as a parameter. Test our solution. Submit it as a Linux patch. Don't make any mistakes (test our submission).

1.2 Application

We began by reading the textbook Linux Device Drivers, Third Edition. Chapter 16 guided us towards modifying the sbull.c file provided by the publisher. We learned how to build and run the module while reading chapter 2. We follow the rules on the chapter 2 and modify the sbull.c file. After that, we set the driver to build as module by typing make menuconfig and build the kernel to generate the rd2.ko. We scp the rd2.ko into QEMU and (test)

1.3 Algorithm

Allocate a block of memory. Any time we write into that memory, apply the crypto algorithm and key to the data before writing. Any time we read from that memory, apply the crypto algorithm and key before returning the data.

1.4 Pseudo Codes

```
Allocate block of memory per request
make crypto cipher using key
if writing to memory:
        memset = (start,0,end) //set the memory to the data
        data = encrypt(data)
else:
        data = decrypt(data)
return data
```

2 Q&A

2.1 Questions

2.1.1 What do you think the main point of this assignment is?

The main point of this assignment was to force us to get up close and personal with several new areas of Linux kernel programming: modifying device drivers to fit our needs, introducing applying practical encryption on the block level, and packaging our solution as a Linux module. Each of these tasks had it's own challenges and obstacles to overcome. Each task relied on the previous, meaning that we needed to understand both the steps we had achieved so far, and what we were trying to accomplish with our next step.

2.1.2 How did you personally approach the problem? Design decisions, algorithm, etc.

We began by trying to understand the problem. What was it really asking us to do? How could we accomplish those steps? We leaned heavily on the class textbook, Linux Device Drivers, Third Edition. Chapter 16 led us through writing a device driver based on sbull.c and chapter 2 helped us to understand how to package our solution as a module. Most of the steps in between were solved with a combination of group discussion, moments of inspiration, and plenty of searching the Internet for similar errors. For example, we got stuck for several hours on a compilation error in the textbook's example file sbull.c, before finally realizing it relied on a different version of the kernel's bio.h.

2.1.3 How did you ensure your solution was correct? Testing details, for instance.

We knew that our module should be performing encryption. So how could we test something that should be performing encryption. Make a file and write some data to it. If you can read it back, encryption/decryption might be working. But how can we tell for sure? Simple, just disable our module. If the file we made earlier is now gibberish, we know that it was encrypted, and that we were also able to decrypt it while the module was running.

2.1.4 What did you learn?

We learned to give ourselves plenty of time with these assignments. This one had some time-consuming surprises for us. Luckily, we began working on the assignment right away, so we had enough time to work through each problem. The interesting and challenging thing about these assignments is that they have several components, each of which is new for us. In this example, we faced at least 6 new problems: writing a device driver, adding encryption, testing, bundling our solution as a module, then bringing our module into the VM. Many of these problems were not-insignificant challenges that required research, time, and some trial-and-error to both solve and attempt to understand why our solution worked.

3 Logs

3.1 Work-flow and scheduling

We used a different work-flow for different portions of the assignment. We usually worked in a group of 3, but often one or two of us were researching a problem while the others attempted trial-and-error, or probed the problem in an attempt to better understand the details. When we found a solution and one of us understood it, we tried to explain it both to ourselves and the others. Followed by promptly making a git commit or editing a Makefile to recreate the circumstances we used to solve the problem.

For file organization, we create a folder in our team directory for each assignment. If we are attempting to solve a particular problem in more than one way, we create git branches until someone solves it. Then we merge back to master and proceed to the next problem together.

3.2 Work log

Our team met Monday, Wednesday, and Friday from 2-4 pm, and as the deadline approached we also met over the weekend.

3.3 Git commit log

```
commit 29f3058a70efc8de21af6ab6b90c887afbc0e480
Author: Levi Willmeth <levi.willmeth@gmail.com>
Date:   2017-05-22

    Update Kconfig and Makefile to working versions

commit b3b0c6db967cc7ae85ed892111fd05782429b735
Author: Levi Willmeth <levi.willmeth@gmail.com>
Date:   2017-05-22

    Add rd2 files, update Makefile and Kconfig

commit acf53576d3fc0a25128e9f76de41dbda9452b918
Author: Levi Willmeth <levi.willmeth@gmail.com>
Date:   2017-05-22

    Merge in testing branch

commit 0ae768ede536e97d9134e81f439a103411e6648c
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15

    delete unnessary file

commit c5b6ba77d07e996be3b7cbcfcb1272aa8cac283b
Merge: 72a4443 5629be4
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15

    Merge branch 'master' of https://github.com/BeNsAeI/CS444

commit 5629be4e2a8ed0d839c7dbe204968e67a2947d1b
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15

    delete the file

commit 51d07eec0d70d2b644e2bb8806aa9e70e09dac70
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15

    upload the file need to change

commit 490998a4ae14dadded40e12bdf0dc5fccff8060b
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15

    upload requirement of hw3

commit 72a4443cd53ea10763b755eb35f566c40f2b2b34
Author: wangdaye123 <wangzhao@oregonstate.edu>
Date:   2017-05-15
```

```
    add hw3 requirement and file need to change

commit ef3752d958a125b8f5db236a8ed812c6def6ce08
Author: BeNsAeI <behnam.saeidi@gmail.com>
Date:   2017-05-10

    added sstf.cfile

commit f660de28c8c30bb3bf558dffae2ceae88c19fbe6
Author: Behnam Saeedi <saeedib@os-class.engr.oregonstate.edu>
Date:   2017-05-10

    backup files

commit 53780e85fd233b72b073bdea2caf50cbcd28d5ea
Author: BeNsAeI <behnam.saeidi@gmail.com>
Date:   2017-05-09

    added lecture notes for 9th lecture
```

4 Code

4.1 rd2.c

```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>

#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/vmalloc.h>
#include <linux/genhd.h>
#include <linux/blkdev.h>
#include <linux/hdreg.h>

#include <linux/crypto.h>

MODULE_LICENSE("Dual BSD/GPL");
static char *Version = "1.4";


static struct crypto_cipher *cipher;

static int major_num = 0;
module_param(major_num, int, 0);
static int logical_block_size = 512;
module_param(logical_block_size, int, 0);
static int nsectors = 1024;
module_param(nsectors, int, 0);

#define KEY_SZ 8
```

```
static char* key = "privateKey";
module_param(key, charp, 0);
#define KERNEL_SECTOR_SIZE 512

static struct request_queue *Queue;

static struct rd2_device {
unsigned long size;
spinlock_t lock;
u8 *data;
struct gendisk *gd;
} Device;

static void rd2_transfer(struct rd2_device *dev, sector_t sector,
unsigned long nsect, char *buffer, int write) {
unsigned long offset = sector * logical_block_size;
unsigned long nbytes = nsect * logical_block_size;
unsigned int i;

//printk("[KERNEL DEBUG] rd2: transfer_key > %s\n",key);

if ((offset + nbytes) > dev->size) {
printk (KERN_NOTICE "[KERNEL DEBUG] rd2(%s): Beyond-end write (%ld %ld)\n", Version, offset, nb
return;
}

crypto_cipher_clear_flags(cipher, ~0);
crypto_cipher_setkey(cipher, key, strlen(key));

if(write) {
for(i = 0; i < nbytes; i += crypto_cipher_blocksize(cipher)) {
memset(dev->data + offset + i, 0, crypto_cipher_blocksize(cipher));
crypto_cipher_encrypt_one(cipher, dev->data + offset + i, buffer + i);
}
} else {
for(i = 0; i < nbytes; i += crypto_cipher_blocksize(cipher)) {
crypto_cipher_decrypt_one(cipher, buffer + i, dev->data + offset + i);
}
}
}

static void rd2_request(struct request_queue *q) {
struct request *req;

req = blk_fetch_request(q);
while (req != NULL) {
if (req == NULL || (req->cmd_type != REQ_TYPE_FS)) {
__blk_end_request_all(req, -EIO);
continue;
}
rd2_transfer(&Device, blk_rq_pos(req), blk_rq_cur_sectors(req),
bio_data(req->bio), rq_data_dir(req));

if ( ! __blk_end_request_cur(req, 0) ) {
req = blk_fetch_request(q);
```

```
}
}
}

int rd2_getgeo(struct block_device * block_device, struct hd_geometry * geo) {
long size;

size = Device.size * (logical_block_size / KERNEL_SECTOR_SIZE);
geo->cylinders = (size & ~0x3f) >> 6;
geo->heads = 4;
geo->sectors = 16;
geo->start = 0;
return 0;
}

static struct block_device_operations rd2_ops = {
.owner  = THIS_MODULE,
.getgeo = rd2_getgeo
};

static int __init rd2_init(void) {
Device.size = nsectors * logical_block_size;
spin_lock_init(&Device.lock);
Device.data = vmalloc(Device.size);
if (Device.data == NULL)
return -ENOMEM;
/*
* Get a request queue.
*/
Queue = blk_init_queue(rd2_request, &Device.lock);
if (Queue == NULL)
goto out;
blk_queue_logical_block_size(Queue, logical_block_size);
/*
* Get registered.
*/
major_num = register_blkdev(major_num, "rd2");
if (major_num < 0) {
goto out;
}

/* allocing cipher */
cipher = crypto_alloc_cipher("aes",0,0);
if(!cipher){
goto out;
}

printk("[KERNEL DEBUG] rd2: enc_key > %s\n",key);
crypto_cipher_setkey(cipher,key,strlen(key));

/*
* And the gendisk structure.
*/
Device.gd = alloc_disk(16);
if (!Device.gd)
```

```
goto out_unregister;
Device.gd->major = major_num;
Device.gd->first_minor = 0;
Device.gd->fops = &rd2_ops;
Device.gd->private_data = &Device;
strcpy(Device.gd->disk_name, "rd20");
set_capacity(Device.gd, nsectors);
Device.gd->queue = Queue;
add_disk(Device.gd);
return 0;

out_unregister:
unregister_blkdev(major_num, "rd2");
out:
vfree(Device.data);
return -ENOMEM;
}

static void __exit rd2_exit(void){

crypto_free_cipher(cipher);

del_gendisk(Device.gd);
put_disk(Device.gd);
unregister_blkdev(major_num, "rd2");
blk_cleanup_queue(Queue);
}

module_init(rd2_init);
module_exit(rd2_exit);
```

4.2 Kconfig

```
#
# Block device driver configuration
#

menuconfig BLK_DEV
        bool "Block devices"
        depends on BLOCK
        default y
        ---help---
          Say Y here to get to see options for various different block device
          drivers. This option alone does not add any kernel code.

          If you say N, all options in this submenu will be skipped and disabled;
          only do this if you know what you are doing.

if BLK_DEV

config BLK_DEV_RD2
        tristate "RD2 DRIVER TEST 10-03"

config BLK_DEV_NULL_BLK
```

```
        tristate "Null test block driver"

config BLK_DEV_FD
        tristate "Normal floppy disk support"
        depends on ARCH_MAY_HAVE_PC_FDC
        ---help---
          If you want to use the floppy disk drive(s) of your PC under Linux,
          say Y. Information about this driver, especially important for IBM
          Thinkpad users, is contained in
          <file:Documentation/blockdev/floppy.txt>.
          That file also contains the location of the Floppy driver FAQ as
          well as location of the fdutils package used to configure additional
          parameters of the driver at run time.

          To compile this driver as a module, choose M here: the
          module will be called floppy.

config AMIGA_FLOPPY
        tristate "Amiga floppy support"
        depends on AMIGA

config ATARI_FLOPPY
        tristate "Atari floppy support"
        depends on ATARI

config MAC_FLOPPY
        tristate "Support for PowerMac floppy"
        depends on PPC_PMAC && !PPC_PMAC64
        help
          If you have a SWIM-3 (Super Woz Integrated Machine 3; from Apple)
          floppy controller, say Y here. Most commonly found in PowerMacs.

config BLK_DEV_SWIM
        tristate "Support for SWIM Macintosh floppy"
        depends on M68K && MAC
        help
          You should select this option if you want floppy support
          and you don't have a II, IIfx, Q900, Q950 or AV series.

config AMIGA_Z2RAM
        tristate "Amiga Zorro II ramdisk support"
        depends on ZORRO
        help
          This enables support for using Chip RAM and Zorro II RAM as a
          ramdisk or as a swap partition. Say Y if you want to include this
          driver in the kernel.

          To compile this driver as a module, choose M here: the
          module will be called z2ram.

config GDROM
        tristate "SEGA Dreamcast GD-ROM drive"
        depends on SH_DREAMCAST
        help
          A standard SEGA Dreamcast comes with a modified CD ROM drive called a
```

```
          "GD-ROM" by SEGA to signify it is capable of reading special disks
          with up to 1 GB of data. This drive will also read standard CD ROM
          disks. Select this option to access any disks in your GD ROM drive.
          Most users will want to say "Y" here.
          You can also build this as a module which will be called gdrom.

config PARIDE
        tristate "Parallel port IDE device support"
        depends on PARPORT_PC
        ---help---
          There are many external CD-ROM and disk devices that connect through
          your computer's parallel port. Most of them are actually IDE devices
          using a parallel port IDE adapter. This option enables the PARIDE
          subsystem which contains drivers for many of these external drives.
          Read <file:Documentation/blockdev/paride.txt> for more information.

          If you have said Y to the "Parallel-port support" configuration
          option, you may share a single port between your printer and other
          parallel port devices. Answer Y to build PARIDE support into your
          kernel, or M if you would like to build it as a loadable module. If
          your parallel port support is in a loadable module, you must build
          PARIDE as a module. If you built PARIDE support into your kernel,
          you may still build the individual protocol modules and high-level
          drivers as loadable modules. If you build this support as a module,
          it will be called paride.

          To use the PARIDE support, you must say Y or M here and also to at
          least one high-level driver (e.g. "Parallel port IDE disks",
          "Parallel port ATAPI CD-ROMs", "Parallel port ATAPI disks" etc.) and
          to at least one protocol driver (e.g. "ATEN EH-100 protocol",
          "MicroSolutions backpack protocol", "DataStor Commuter protocol"
          etc.).

source "drivers/block/paride/Kconfig"

source "drivers/block/mtip32xx/Kconfig"

source "drivers/block/zram/Kconfig"

config BLK_CPQ_DA
        tristate "Compaq SMART2 support"
        depends on PCI && VIRT_TO_BUS && 0
        help
          This is the driver for Compaq Smart Array controllers.  Everyone
          using these boards should say Y here.  See the file
          <file:Documentation/blockdev/cpqarray.txt> for the current list of
          boards supported by this driver, and for further information on the
          use of this driver.

config BLK_CPQ_CISS_DA
        tristate "Compaq Smart Array 5xxx support"
        depends on PCI
        select CHECK_SIGNATURE
        help
          This is the driver for Compaq Smart Array 5xxx controllers.
```

```
          Everyone using these boards should say Y here.
          See <file:Documentation/blockdev/cciss.txt> for the current list of
          boards supported by this driver, and for further information
          on the use of this driver.

config CISS_SCSI_TAPE
        bool "SCSI tape drive support for Smart Array 5xxx"
        depends on BLK_CPQ_CISS_DA && PROC_FS
        depends on SCSI=y || SCSI=BLK_CPQ_CISS_DA
        help
          When enabled (Y), this option allows SCSI tape drives and SCSI medium
          changers (tape robots) to be accessed via a Compaq 5xxx array
          controller.  (See <file:Documentation/blockdev/cciss.txt> for more details.)

          "SCSI support" and "SCSI tape support" must also be enabled for this
          option to work.

          When this option is disabled (N), the SCSI portion of the driver
          is not compiled.

config BLK_DEV_DAC960
        tristate "Mylex DAC960/DAC1100 PCI RAID Controller support"
        depends on PCI
        help
          This driver adds support for the Mylex DAC960, AcceleRAID, and
          eXtremeRAID PCI RAID controllers.  See the file
          <file:Documentation/blockdev/README.DAC960> for further information
          about this driver.

          To compile this driver as a module, choose M here: the
          module will be called DAC960.

config BLK_DEV_UMEM
        tristate "Micro Memory MM5415 Battery Backed RAM support"
        depends on PCI
        ---help---
          Saying Y here will include support for the MM5415 family of
          battery backed (Non-volatile) RAM cards.
          <http://www.umem.com/>

          The cards appear as block devices that can be partitioned into
          as many as 15 partitions.

          To compile this driver as a module, choose M here: the
          module will be called umem.

          The umem driver has not yet been allocated a MAJOR number, so
          one is chosen dynamically.

config BLK_DEV_UBD
        bool "Virtual block device"
        depends on UML
        ---help---
          The User-Mode Linux port includes a driver called UBD which will let
          you access arbitrary files on the host computer as block devices.
```

```
          Unless you know that you do not need such virtual block devices say
          Y here.

config BLK_DEV_UBD_SYNC
        bool "Always do synchronous disk IO for UBD"
        depends on BLK_DEV_UBD
        ---help---
          Writes to the virtual block device are not immediately written to the
          host's disk; this may cause problems if, for example, the User-Mode
          Linux 'Virtual Machine' uses a journalling filesystem and the host
          computer crashes.

          Synchronous operation (i.e. always writing data to the host's disk
          immediately) is configurable on a per-UBD basis by using a special
          kernel command line option.  Alternatively, you can say Y here to
          turn on synchronous operation by default for all block devices.

          If you're running a journalling file system (like reiserfs, for
          example) in your virtual machine, you will want to say Y here.  If
          you care for the safety of the data in your virtual machine, Y is a
          wise choice too.  In all other cases (for example, if you're just
          playing around with User-Mode Linux) you can choose N.

config BLK_DEV_COW_COMMON
        bool
        default BLK_DEV_UBD

config BLK_DEV_LOOP
        tristate "Loopback device support"
        ---help---
          Saying Y here will allow you to use a regular file as a block
          device; you can then create a file system on that block device and
          mount it just as you would mount other block devices such as hard
          drive partitions, CD-ROM drives or floppy drives. The loop devices
          are block special device files with major number 7 and typically
          called /dev/loop0, /dev/loop1 etc.

          This is useful if you want to check an ISO 9660 file system before
          burning the CD, or if you want to use floppy images without first
          writing them to floppy. Furthermore, some Linux distributions avoid
          the need for a dedicated Linux partition by keeping their complete
          root file system inside a DOS FAT file using this loop device
          driver.

          To use the loop device, you need the losetup utility, found in the
          util-linux package, see
          <ftp://ftp.kernel.org/pub/linux/utils/util-linux/>.

          The loop device driver can also be used to "hide" a file system in
          a disk partition, floppy, or regular file, either using encryption
          (scrambling the data) or steganography (hiding the data in the low
          bits of, say, a sound file). This is also safe if the file resides
          on a remote file server.

          There are several ways of encrypting disks. Some of these require
```

```
          kernel patches. The vanilla kernel offers the cryptoloop option
          and a Device Mapper target (which is superior, as it supports all
          file systems). If you want to use the cryptoloop, say Y to both
          LOOP and CRYPTOLOOP, and make sure you have a recent (version 2.12
          or later) version of util-linux. Additionally, be aware that
          the cryptoloop is not safe for storing journaled filesystems.

          Note that this loop device has nothing to do with the loopback
          device used for network connections from the machine to itself.

          To compile this driver as a module, choose M here: the
          module will be called loop.

          Most users will answer N here.

config BLK_DEV_LOOP_MIN_COUNT
        int "Number of loop devices to pre-create at init time"
        depends on BLK_DEV_LOOP
        default 8
        help
          Static number of loop devices to be unconditionally pre-created
          at init time.

          This default value can be overwritten on the kernel command
          line or with module-parameter loop.max_loop.

          The historic default is 8. If a late 2011 version of losetup(8)
          is used, it can be set to 0, since needed loop devices can be
          dynamically allocated with the /dev/loop-control interface.

config BLK_DEV_CRYPTOLOOP
        tristate "Cryptoloop Support"
        select CRYPTO
        select CRYPTO_CBC
        depends on BLK_DEV_LOOP
        ---help---
          Say Y here if you want to be able to use the ciphers that are
          provided by the CryptoAPI as loop transformation. This might be
          used as hard disk encryption.

          WARNING: This device is not safe for journaled file systems like
          ext3 or Reiserfs. Please use the Device Mapper crypto module
          instead, which can be configured to be on-disk compatible with the
          cryptoloop device.

source "drivers/block/drbd/Kconfig"

config BLK_DEV_NBD
        tristate "Network block device support"
        depends on NET
        ---help---
          Saying Y here will allow your computer to be a client for network
          block devices, i.e. it will be able to use block devices exported by
          servers (mount file systems on them etc.). Communication between
          client and server works over TCP/IP networking, but to the client
```

```
          program this is hidden: it looks like a regular local file access to
          a block device special file such as /dev/nd0.

          Network block devices also allows you to run a block-device in
          userland (making server and client physically the same computer,
          communicating using the loopback network device).

          Read <file:Documentation/blockdev/nbd.txt> for more information,
          especially about where to find the server code, which runs in user
          space and does not need special kernel support.

          Note that this has nothing to do with the network file systems NFS
          or Coda; you can say N here even if you intend to use NFS or Coda.

          To compile this driver as a module, choose M here: the
          module will be called nbd.

          If unsure, say N.

config BLK_DEV_NVME
        tristate "NVM Express block device"
        depends on PCI
        ---help---
          The NVM Express driver is for solid state drives directly
          connected to the PCI or PCI Express bus.  If you know you
          don't have one of these, it is safe to answer N.

          To compile this driver as a module, choose M here: the
          module will be called nvme.

config BLK_DEV_SKD
        tristate "STEC S1120 Block Driver"
        depends on PCI
        depends on 64BIT
        ---help---
        Saying Y or M here will enable support for the
        STEC, Inc. S1120 PCIe SSD.

        Use device /dev/skd$N amd /dev/skd$Np$M.

config BLK_DEV_OSD
        tristate "OSD object-as-blkdev support"
        depends on SCSI_OSD_ULD
        ---help---
          Saying Y or M here will allow the exporting of a single SCSI
          OSD (object-based storage) object as a Linux block device.

          For example, if you create a 2G object on an OSD device,
          you can then use this module to present that 2G object as
          a Linux block device.

          To compile this driver as a module, choose M here: the
          module will be called osdblk.

          If unsure, say N.
```

```
config BLK_DEV_SX8
        tristate "Promise SATA SX8 support"
        depends on PCI
        ---help---
          Saying Y or M here will enable support for the
          Promise SATA SX8 controllers.

          Use devices /dev/sx8/$N and /dev/sx8/$Np$M.

config BLK_DEV_RAM
        tristate "RAM block device support"
        ---help---
          Saying Y here will allow you to use a portion of your RAM memory as
          a block device, so that you can make file systems on it, read and
          write to it and do all the other things that you can do with normal
          block devices (such as hard drives). It is usually used to load and
          store a copy of a minimal root file system off of a floppy into RAM
          during the initial install of Linux.

          Note that the kernel command line option "ramdisk=XX" is now obsolete.
          For details, read <file:Documentation/blockdev/ramdisk.txt>.

          To compile this driver as a module, choose M here: the
          module will be called brd. An alias "rd" has been defined
          for historical reasons.

          Most normal users won't need the RAM disk functionality, and can
          thus say N here.

config BLK_DEV_RAM_COUNT
        int "Default number of RAM disks"
        default "16"
        depends on BLK_DEV_RAM
        help
          The default value is 16 RAM disks. Change this if you know what you
          are doing. If you boot from a filesystem that needs to be extracted
          in memory, you will need at least one RAM disk (e.g. root on cramfs).

config BLK_DEV_RAM_SIZE
        int "Default RAM disk size (kbytes)"
        depends on BLK_DEV_RAM
        default "4096"
        help
          The default value is 4096 kilobytes. Only change this if you know
          what you are doing.

config BLK_DEV_XIP
        bool "Support XIP filesystems on RAM block device"
        depends on BLK_DEV_RAM
        default n
        help
          Support XIP filesystems (such as ext2 with XIP support on) on
          top of block ram device. This will slightly enlarge the kernel, and
          will prevent RAM block device backing store memory from being
```

```
          allocated from highmem (only a problem for highmem systems).

config CDROM_PKTCDVD
        tristate "Packet writing on CD/DVD media"
        depends on !UML
        help
          If you have a CDROM/DVD drive that supports packet writing, say
          Y to include support. It should work with any MMC/Mt Fuji
          compliant ATAPI or SCSI drive, which is just about any newer
          DVD/CD writer.

          Currently only writing to CD-RW, DVD-RW, DVD+RW and DVDRAM discs
          is possible.
          DVD-RW disks must be in restricted overwrite mode.

          See the file <file:Documentation/cdrom/packet-writing.txt>
          for further information on the use of this driver.

          To compile this driver as a module, choose M here: the
          module will be called pktcdvd.

config CDROM_PKTCDVD_BUFFERS
        int "Free buffers for data gathering"
        depends on CDROM_PKTCDVD
        default "8"
        help
          This controls the maximum number of active concurrent packets. More
          concurrent packets can increase write performance, but also require
          more memory. Each concurrent packet will require approximately 64Kb
          of non-swappable kernel memory, memory which will be allocated when
          a disc is opened for writing.

config CDROM_PKTCDVD_WCACHE
        bool "Enable write caching"
        depends on CDROM_PKTCDVD
        help
          If enabled, write caching will be set for the CD-R/W device. For now
          this option is dangerous unless the CD-RW media is known good, as we
          don't do deferred write error handling yet.

config ATA_OVER_ETH
        tristate "ATA over Ethernet support"
        depends on NET
        help
        This driver provides Support for ATA over Ethernet block
        devices like the Coraid EtherDrive (R) Storage Blade.

config MG_DISK
        tristate "mGine mflash, gflash support"
        depends on ARM && GPIOLIB
        help
          mGine mFlash(gFlash) block device driver

config MG_DISK_RES
        int "Size of reserved area before MBR"
```

```
        depends on MG_DISK
        default 0
        help
          Define size of reserved area that usually used for boot. Unit is KB.
          All of the block device operation will be taken this value as start
          offset
          Examples:
                       1024 => 1 MB

config SUNVDC
        tristate "Sun Virtual Disk Client support"
        depends on SUN_LDOMS
        help
          Support for virtual disk devices as a client under Sun
          Logical Domains.

source "drivers/s390/block/Kconfig"

config XILINX_SYSACE
        tristate "Xilinx SystemACE support"
        depends on 4xx || MICROBLAZE
        help
          Include support for the Xilinx SystemACE CompactFlash interface

config XEN_BLKDEV_FRONTEND
        tristate "Xen virtual block device support"
        depends on XEN
        default y
        select XEN_XENBUS_FRONTEND
        help
          This driver implements the front-end of the Xen virtual
          block device driver.  It communicates with a back-end driver
          in another domain which drives the actual block device.

config XEN_BLKDEV_BACKEND
        tristate "Xen block-device backend driver"
        depends on XEN_BACKEND
        help
          The block-device backend driver allows the kernel to export its
          block devices to other guests via a high-performance shared-memory
          interface.

          The corresponding Linux frontend driver is enabled by the
          CONFIG_XEN_BLKDEV_FRONTEND configuration option.

          The backend driver attaches itself to a any block device specified
          in the XenBus configuration. There are no limits to what the block
          device as long as it has a major and minor.

          If you are compiling a kernel to run in a Xen block backend driver
          domain (often this is domain 0) you should say Y here. To
          compile this driver as a module, chose M here: the module
          will be called xen-blkback.
```

```
config VIRTIO_BLK
        tristate "Virtio block driver"
        depends on VIRTIO
        ---help---
          This is the virtual block driver for virtio.  It can be used with
          lguest or QEMU based VMMs (like KVM or Xen).  Say Y or M.

config BLK_DEV_HD
        bool "Very old hard disk (MFM/RLL/IDE) driver"
        depends on HAVE_IDE
        depends on !ARM || ARCH_RPC || BROKEN
        help
          This is a very old hard disk driver that lacks the enhanced
          functionality of the newer ones.

          It is required for systems with ancient MFM/RLL/ESDI drives.

          If unsure, say N.

config BLK_DEV_RBD
        tristate "Rados block device (RBD)"
        depends on INET && BLOCK
        select CEPH_LIB
        select LIBCRC32C
        select CRYPTO_AES
        select CRYPTO
        default n
        help
          Say Y here if you want include the Rados block device, which stripes
          a block device over objects stored in the Ceph distributed object
          store.

          More information at http://ceph.newdream.net/.

          If unsure, say N.

config BLK_DEV_RSXX
        tristate "IBM Flash Adapter 900GB Full Height PCIe Device Driver"
        depends on PCI
        help
          Device driver for IBM's high speed PCIe SSD
          storage device: Flash Adapter 900GB Full Height.

          To compile this driver as a module, choose M here: the
          module will be called rsxx.

endif # BLK_DEV
```

4.3 Makefile

```
#
# Makefile for the kernel block device drivers.
#
# 12 June 2000, Christoph Hellwig <hch@infradead.org>
```

```
# Rewritten to use lists instead of if-statements.
#

obj-$(CONFIG_MAC_FLOPPY)        += swim3.o
obj-$(CONFIG_BLK_DEV_SWIM)      += swim_mod.o
obj-$(CONFIG_BLK_DEV_FD)        += floppy.o
obj-$(CONFIG_AMIGA_FLOPPY)      += amiflop.o
obj-$(CONFIG_PS3_DISK)          += ps3disk.o
obj-$(CONFIG_PS3_VRAM)          += ps3vram.o
obj-$(CONFIG_ATARI_FLOPPY)      += ataflop.o
obj-$(CONFIG_AMIGA_Z2RAM)       += z2ram.o
obj-$(CONFIG_BLK_DEV_RAM)       += brd.o
obj-$(CONFIG_BLK_DEV_LOOP)      += loop.o
obj-$(CONFIG_BLK_CPQ_DA)        += cpqarray.o
obj-$(CONFIG_BLK_CPQ_CISS_DA)  += cciss.o
obj-$(CONFIG_BLK_DEV_DAC960)    += DAC960.o
obj-$(CONFIG_XILINX_SYSACE)     += xsysace.o
obj-$(CONFIG_CDROM_PKTCDVD)     += pktcdvd.o
obj-$(CONFIG_MG_DISK)           += mg_disk.o
obj-$(CONFIG_SUNVDC)            += sunvdc.o
obj-$(CONFIG_BLK_DEV_NVME)      += nvme.o
obj-$(CONFIG_BLK_DEV_SKD)       += skd.o
obj-$(CONFIG_BLK_DEV_OSD)       += osdblk.o

obj-$(CONFIG_BLK_DEV_UMEM)      += umem.o
obj-$(CONFIG_BLK_DEV_NBD)       += nbd.o
obj-$(CONFIG_BLK_DEV_CRYPTOLOOP) += cryptoloop.o
obj-$(CONFIG_VIRTIO_BLK)        += virtio_blk.o

obj-$(CONFIG_BLK_DEV_SX8)       += sx8.o
obj-$(CONFIG_BLK_DEV_HD)        += hd.o

obj-$(CONFIG_XEN_BLKDEV_FRONTEND)       += xen-blkfront.o
obj-$(CONFIG_XEN_BLKDEV_BACKEND)        += xen-blkback/
obj-$(CONFIG_BLK_DEV_DRBD)     += drbd/
obj-$(CONFIG_BLK_DEV_RBD)     += rbd.o
obj-$(CONFIG_BLK_DEV_PCIESSD_MTIP32XX)  += mtip32xx/

obj-$(CONFIG_BLK_DEV_RSXX) += rsxx/
obj-$(CONFIG_BLK_DEV_NULL_BLK)  += null_blk.o
obj-$(CONFIG_ZRAM) += zram/
obj-$(CONFIG_BLK_DEV_RD2) += rd2.o

nvme-y          := nvme-core.o nvme-scsi.o
skd-y           := skd_main.o
swim_mod-y      := swim.o swim_asm.o
```

4.4 Patch