SHAP review meeting

DRBD and coccinelle

02/12/2019

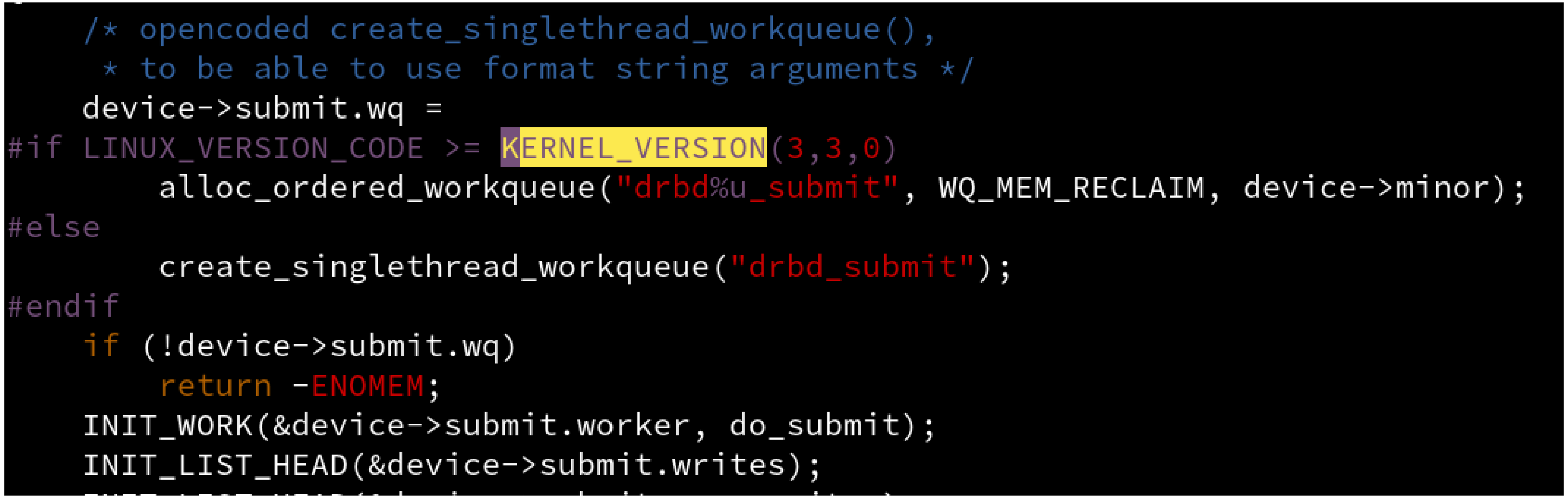
What is coccinelle

- Tool for pattern matching and text transformation that has many uses in kernel development
- The semantic patches detection of problematic programming patterns

Why DRBD need?

- kernel’s interface change due to hardware improvements, driven by general evolution of the code base, etc…
- In tree drivers is not a big issue since “tree wide changes”
- DRBD in “drivers/block/drbd” is 8.4.X. Tumbleweed is now in “drbd-9.0.20”. So DRBD server as KMP in SLE
  OBS test project against openSUSE:Factory is “Kernel:HEAD:KMP/drbd”
- A P1 issue will be filed if failure in SLE due to API change!

builtin KERNEL_VERSION



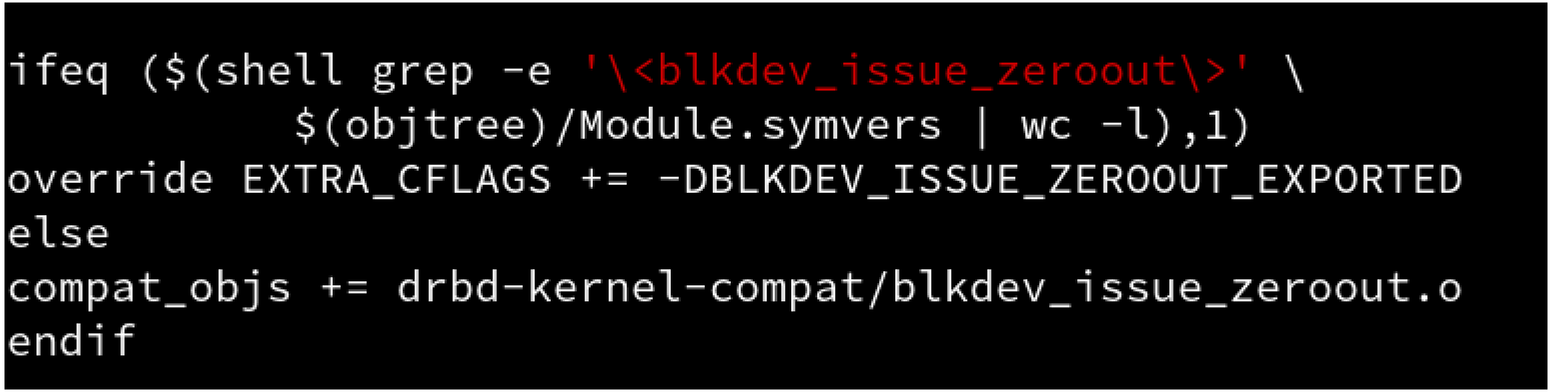
```
    /* opencoded create_singlethread_workqueue(),
     * to be able to use format string arguments */
    device->submit.wq =
#if LINUX_VERSION_CODE >= KERNEL_VERSION(3,3,0)
        alloc_ordered_workqueue("drbd%u_submit", WQ_MEM_RECLAIM, device->minor);
#else
        create_singlethread_workqueue("drbd_submit");
#endif
    if (!device->submit.wq)
        return -ENOMEM;
    INIT_WORK(&device->submit.worker, do_submit);
    INIT_LIST_HEAD(&device->submit.writes);
```

What was in DRBD?

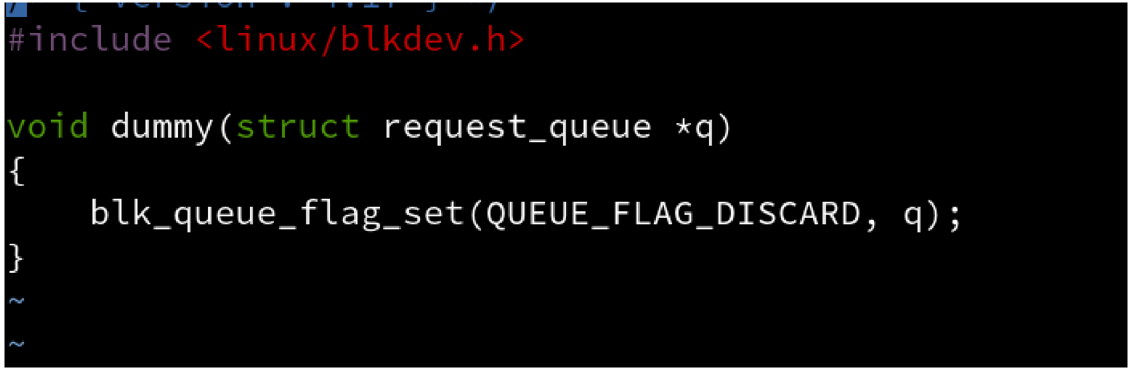
- Detecting the capabilities of the kernel we want to build against.

- The compatibility layer, a huge file containing many #IFDEFs

Detect patch



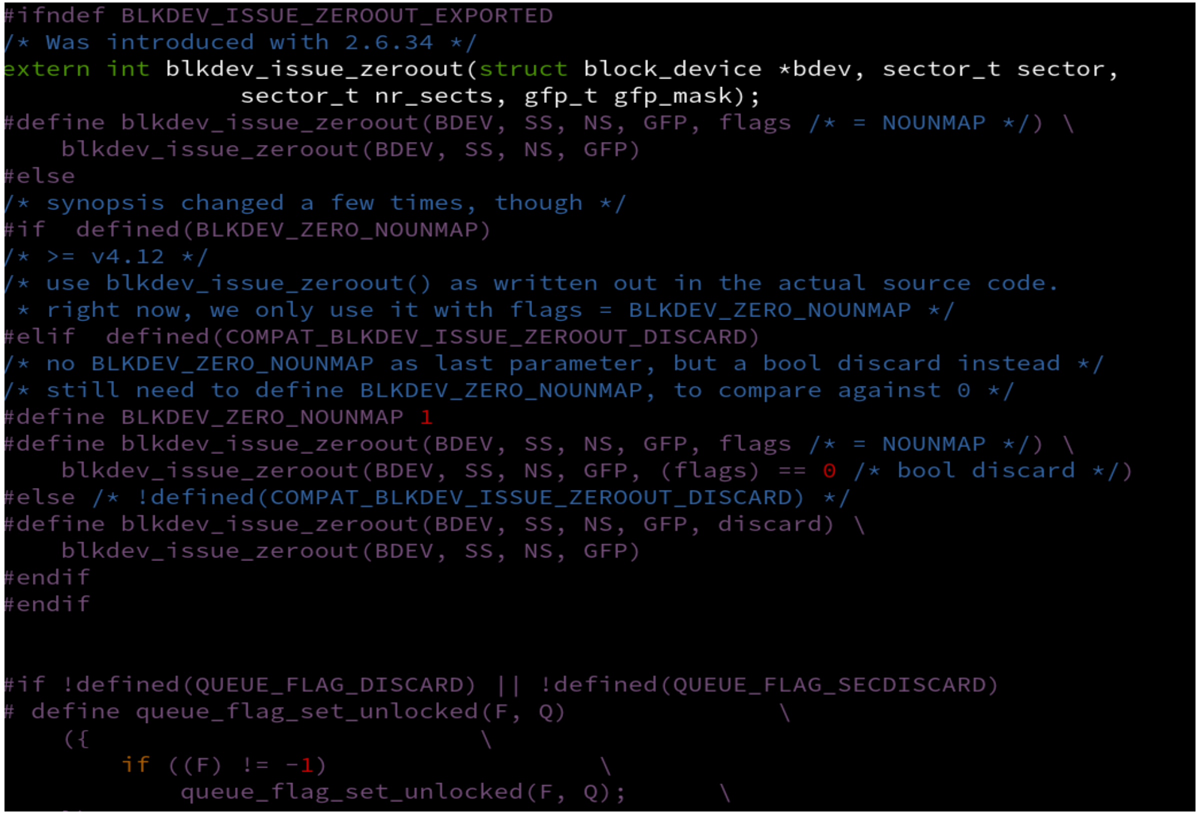
```
ifeq ($(shell grep -e '\<blkdev_issue_zeroout\>' \
           $(objtree)/Module.symvers | wc -l),1)
override EXTRA_CFLAGS += -DBLKDEV_ISSUE_ZEROOUT_EXPORTED
else
compat_objs += drbd-kernel-compat/blkdev_issue_zeroout.o
endif
```



```
#include <linux/blkdev.h>

void dummy(struct request_queue *q)
{
    blk_queue_flag_set(QUEUE_FLAG_DISCARD, q);
}
~
~
```

Compatibility layer



```
#ifndef BLKDEV_ISSUE_ZEROOUT_EXPORTED
/* Was introduced with 2.6.34 */
extern int blkdev_issue_zeroout(struct block_device *bdev, sector_t sector,
                sector_t nr_sects, gfp_t gfp_mask);
#define blkdev_issue_zeroout(BDEV, SS, NS, GFP, flags /* = NOUNMAP */) \
    blkdev_issue_zeroout(BDEV, SS, NS, GFP)
#else
/* synopsis changed a few times, though */
#if  defined(BLKDEV_ZERO_NOUNMAP)
/* >= v4.12 */
/* use blkdev_issue_zeroout() as written out in the actual source code.
 * right now, we only use it with flags = BLKDEV_ZERO_NOUNMAP */
#elif  defined(COMPAT_BLKDEV_ISSUE_ZEROOUT_DISCARD)
/* no BLKDEV_ZERO_NOUNMAP as last parameter, but a bool discard instead */
/* still need to define BLKDEV_ZERO_NOUNMAP, to compare against 0 */
#define BLKDEV_ZERO_NOUNMAP 1
#define blkdev_issue_zeroout(BDEV, SS, NS, GFP, flags /* = NOUNMAP */) \
    blkdev_issue_zeroout(BDEV, SS, NS, GFP, (flags) == 0 /* bool discard */)
#else /* !defined(COMPAT_BLKDEV_ISSUE_ZEROOUT_DISCARD) */
#define blkdev_issue_zeroout(BDEV, SS, NS, GFP, discard) \
    blkdev_issue_zeroout(BDEV, SS, NS, GFP)
#endif
#endif


#if !defined(QUEUE_FLAG_DISCARD) || !defined(QUEUE_FLAG_SECDISCARD)
# define queue_flag_set_unlocked(F, Q)                \
    ({                          \
        if ((F) != -1)                  \
            queue_flag_set_unlocked(F, Q);      \
```

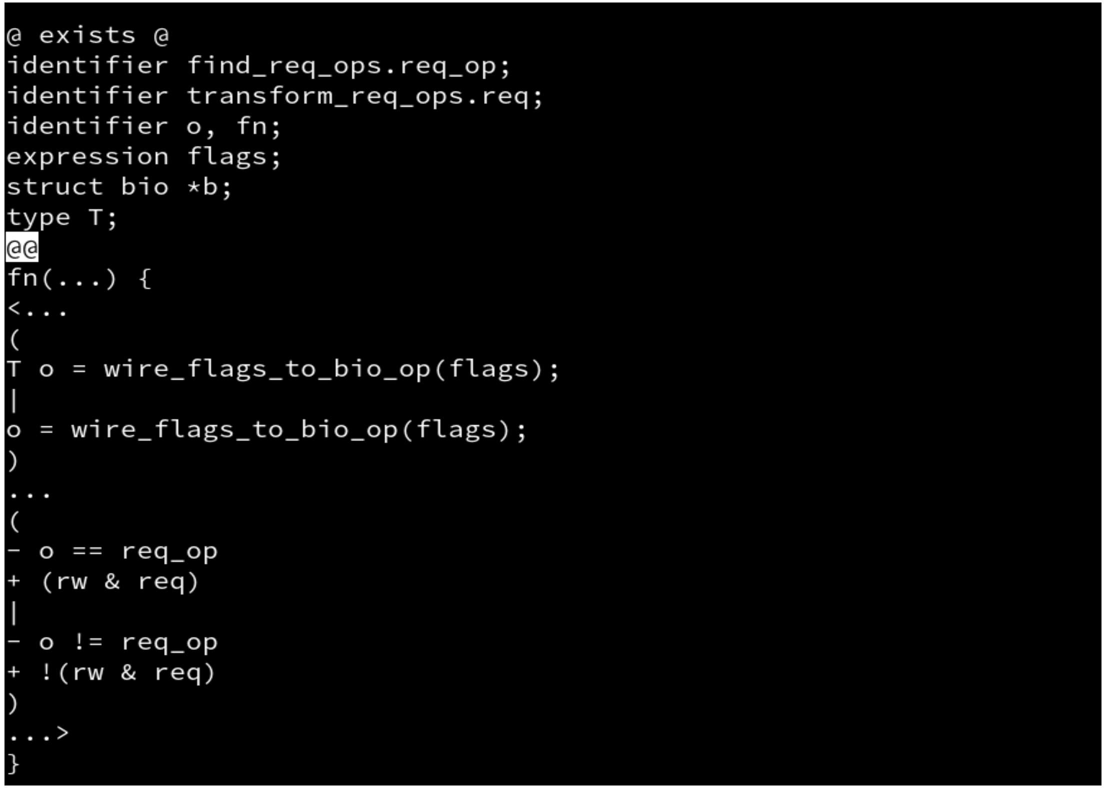
What is in the latest DRBD?

- Detecting the capabilities of the kernel we want to build against.
- Create a compat patch using spatch(from coccinelle)
- Apply the compat patch

Coccinelle patch (example 1)

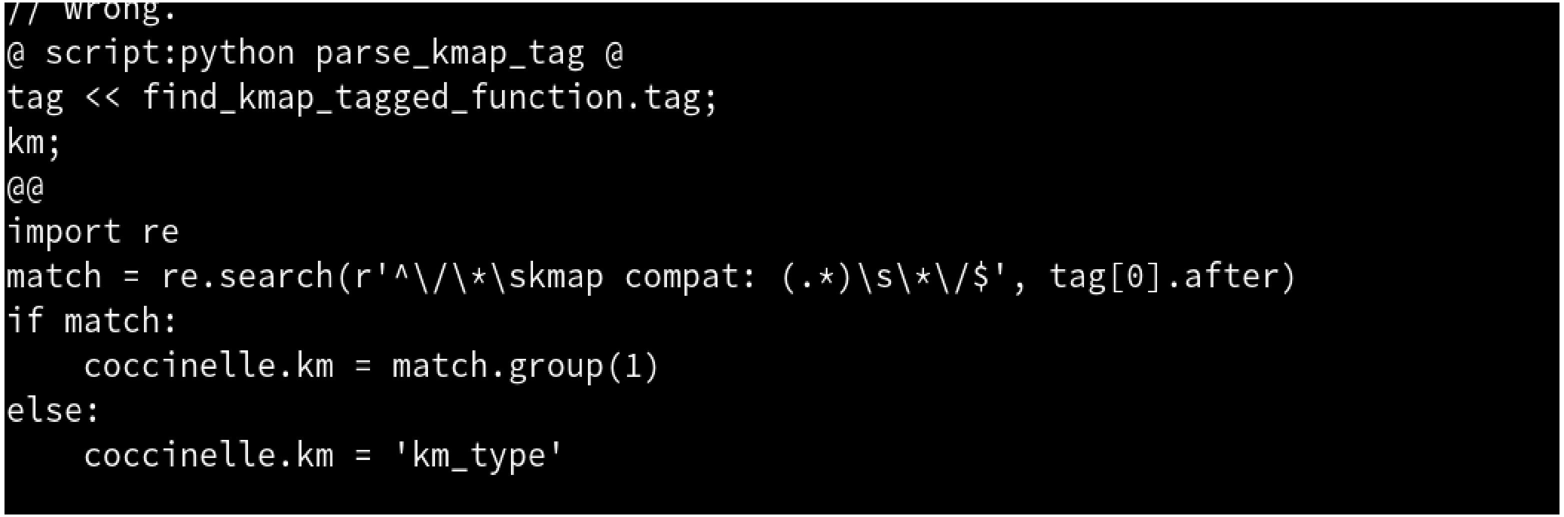
```
// There's some macros defined for debugging (#ifdef BITMAP_DEBUG), and we need
// to change the definition of those as well (and the calls inside them)
@@
identifier device, bitmap_index, start, end, op, buffer;
@@
(
-#define ___bm_op(device, bitmap_index, start, end, op, buffer)
+#define ___bm_op(device, bitmap_index, start, end, op, buffer, km_type)
|
-____bm_op(device, bitmap_index, start, end, op, buffer)
+____bm_op(device, bitmap_index, start, end, op, buffer, km_type)
)
```

Coccinelle patch (example 2)



```
@ exists @
identifier find_req_ops.req_op;
identifier transform_req_ops.req;
identifier o, fn;
expression flags;
struct bio *b;
type T;
@@
fn(...) {
<...
(
T o = wire_flags_to_bio_op(flags);
|
o = wire_flags_to_bio_op(flags);
)
...
(
- o == req_op
+ (rw & req)
|
- o != req_op
+ !(rw & req)
)
...>
}
```

Coccinelle patch (example 3)



```
// wrong:
@ script:python parse_kmap_tag @
tag << find_kmap_tagged_function.tag;
km;
@@
import re
match = re.search(r'^\/\*\skmap compat: (.*)\s\*\/$', tag[0].after)
if match:
    coccinelle.km = match.group(1)
else:
    coccinelle.km = 'km_type'
```

How to get coccinelle env (OCaml)?

- LINBIT server “spatch-as-a-service”. (spatch is the binary of coccinelle)

- Docker container with coccinelle configured available
  https://hub.docker.com/r/linbit/coccinelle

- Install the coccinelle as build require then build DRBD.
  (build DRBD with spatch against a customized patch is not easy...)

Story of build DRBD with spatch

- Detecting the capabilities of current kernel
- Find the needed .cocci patches with binary “gen_patch_names” to generated the “applied_cocci_files”
- Create a “*.compat.cocci”

# Need to accomplish above 3 steps for a customized patch to DRBD

Dependencies of coccinelle

- Packages with ocaml-* as a dependencies hole

  New introduce to SLE15SP2:
  - ocaml-menhir
  - ocaml-parmap

- Potential maintenance issue in future

Dependencies of coccinelle

- Packages with ocaml-* as a dependencies hole

  New introduce to SLE15SP2:
  ocaml-menhir
  ocaml-parmap

- Potential maintenance issue in future

References

- Coccinelle official:
  http://coccinelle.lip6.fr/
- Introduce to kernel development:
  https://lwn.net/Articles/315686/
  https://www.kernel.org/doc/html/v4.11/dev-tools/coccinelle.html
- Coccinelle for newbie:
  https://home.regit.org/technical-articles/coccinelle-for-the-newbie/
- A Linbit blog:
  https://www.linbit.com/en/how-to-make-drbd-compatible-to-the-linux-kernel/

SUSE®

We adapt. You succeed.