technische universität
dortmund

Beastie In For Checkup: Analyzing FreeBSD with LockDoc

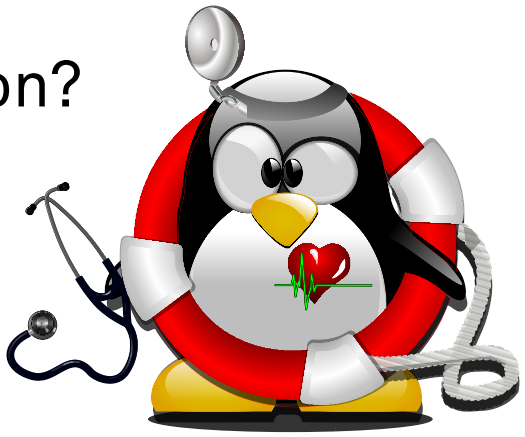
Alexander Lochmann, Horst Schirmeier

alexander.lochmann@tu-dortmund.de
https://ess.cs.tu-dortmund.de/~al

Databases and Information Systems Group
Computer Science 6, TU Dortmund

What is LockDoc?

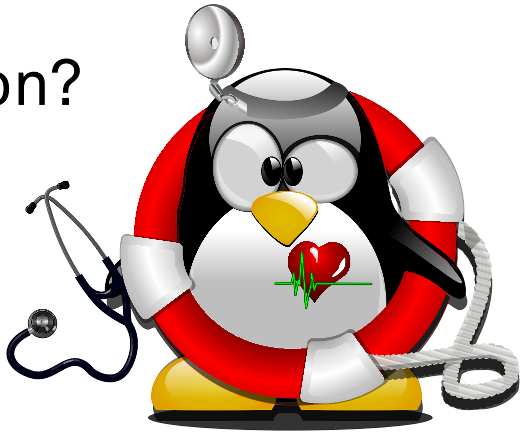
- Tracks **locking pattern** and **data-structure** accesses
- **Recording** performed under a **load**
- Generates documentation, and locates locking bugs
- **Validate** existing locking **documentation**
  - Does the code adhere to the documentation?
- LockDoc study on Linux [3]
  - Validate documentation of 5 data type
  - 53 % of all observed fields accessed consistently with their doc.



Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys'19.

What is LockDoc?

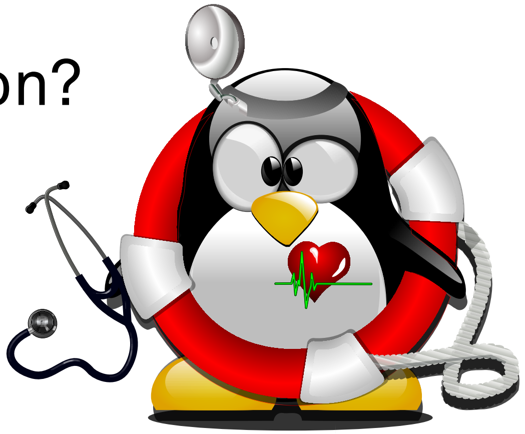
- Tracks **locking pattern** and **data-structure** accesses
- **Recording** performed under a **load**
- Generates documentation, and locates locking bugs
- **Validate** existing locking **documentation**
  - Does the code adhere to the documentation?
- LockDoc study on Linux [3]
  - Validate documentation of 5 data type
  - 53 % of all observed fields accessed consistently with their doc.



- Word-size variables can be accessed without locks
- If no concurrency takes place, no locks needed
- No locks if consistency does not matter

What is LockDoc?

- Tracks **locking pattern** and **data-structure** accesses
- **Recording** performed under a **load**
- Generates documentation, and locates locking bugs
- **Validate** existing locking **documentation**
  - Does the code adhere to the documentation?
- LockDoc study on Linux [3]
  - Validate documentation of 5 data type
  - 53 % of all observed fields accessed consistently with their doc.



→ Real bugs?
→ Issues with LockDoc?

Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys'19.

Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
      away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
      required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
       away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
       required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
      away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
      required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

technische universität
dortmund

Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
      away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
      required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
      away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
      required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

```
struct vnode {
    // […]
    short   v_irflag;/* i frequently read flags */
    seqc_t  v_seqc;  /* i modification count */
    // […]
    /*
     * Filesystem instance stuff
     */
    struct mount *v_mount; /* u [...] */
    TAILQ_ENTRY(vnode) v_nmntvnodes; /* m [...] */
    // [...]
};
```

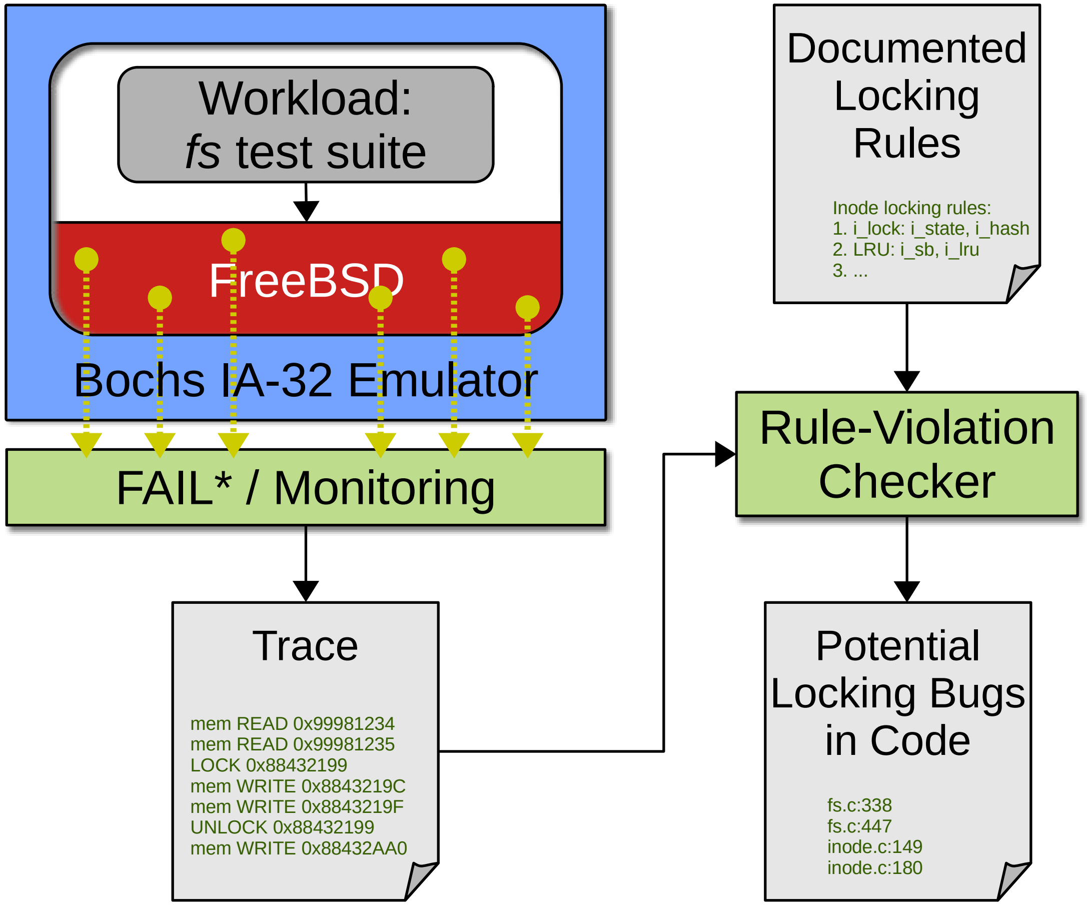
Locking Documentation in FreeBSD

```
/*
 * Reading or writing any of these items requires
   holding the appropriate lock.
 *
 * Lock reference:
 *  c - namecache mutex
 *  i - interlock
 *  l - mp mnt_listmtx or freelist mutex
 *  I - updated with atomics, 0->1 and 1→0
        transitions with interlock held
 *  m - mount point interlock
 *  p - pollinfo lock
 *  u - Only a reference to the vnode is needed to
        read.
 *  v - vnode lock
 *
 * Vnodes may be found on many lists.  The general way
   to deal with operating
 * on a vnode that is on a list is:
 *  1) Lock the list and find the vnode.
 *  2) Lock interlock so that the vnode does not go
       away.
 *  3) Unlock the list to avoid lock order reversals.
 *  4) vget with LK_INTERLOCK and check for ENOENT, or
 *  5) Check for DOOMED if the vnode lock is not
       required.
 *  6) Perform your operation, then vput().
 */
```

sys/sys/vnode.h

```
struct vnode {
    // […]
    short    v_irflag;/* i frequently read flags */
    seqc_t   v_seqc;   /* i modification count */
    // […]
    /*
     * Filesystem instance stuff
     */
    struct mount *v_mount; /* u [...] */
    TAILQ_ENTRY(vnode) v_nmntvnodes; /* m [...] */
    // [...]
};
```

LockDoc – A Different Approach



```
mem READ 0x99981234
mem READ 0x99981235
LOCK 0x88432199
mem WRITE 0x8843219C
mem WRITE 0x8843219F
UNLOCK 0x88432199
mem WRITE 0x88432AA0
```

Inode locking rules:
1. i_lock: i_state, i_hash
2. LRU: i_sb, i_lru
3. ...

```
fs.c:338
fs.c:447
inode.c:149
inode.c:180
```

Instrumentation / Experiment Setup

- i386 FreeBSD 13.0 (Git commit *2134e85bc*)
- Instrument FreeBSD's *Witness* system [2]
  - Uses same lock model as LockDoc: read lock, write lock, rw lock
  - Automatically instrument all lock operations
  - 8 different types recorded: *hardirq*, *lockmgr*, *rm*, *rw*, *sleepable rm*, *sleep mutex*, *spin mutex*, and *sx*
- Using *fs* test suite from *Linux Test Project* as workload
- 26.43 hours runtime (20.22 minutes in a real vm)

FreeBSD Results (1)

- 4 data types: *vnode*, *mount*, *buf*, and *bufobj*
- *bufobj* is embedded in *vnode*

Data Type	#R	#No	#Ob	✓ (%)	~ (%)	✗ (%)
vnode	82	9	73	72.60	27.40	0.00
mount	38	7	31	74.19	25.81	0.00
buf	80	10	70	71.43	27.14	1.43

FreeBSD Results (1)

- 4 data types: *vnode*, *mount*, *buf*, and *bufobj*
- *bufobj* is embedded in *vnode*

Data Type	#R	#No	#Ob	✓ (%)	~ (%)	✗ (%)
vnode	82	9	73	72.60	27.40	0.00
mount	38	7	31	74.19	25.81	0.00
buf	80	10	70	71.43	27.14	$\boxed{1.43}$

Read of
buf.b_error

FreeBSD Results (1)

- 4 data types: *vnode*, *mount*, *buf*, and *bufobj*
- *bufobj* is embedded in *vnode*

Data Type	#R	#No	#Ob	✓ (%)	~ (%)	✗ (%)
vnode	82	9	73	72.60	27.40	0.00
mount	38	7	31	74.19	25.81	0.00
buf	80	10	70	71.43	27.14	1.43

72.41 %

FreeBSD Results (1)

- 4 data types: *vnode*, *mount*, *buf*, and *bufobj*
- *bufobj* is embedded in *vnode*

Data Type	#R	#No	#Ob	✓ (%)	~ (%)	✗ (%)
vnode	82	9	73	72.60	27.40	0.00
mo						.00
buf						.43

→ What about the remaining 27.59 %?

FreeBSD Results (2)

$$0{,}9 \leq s_r < 1$$

[1]https://lists.freebsd.org/archives/freebsd-fs/2021-August/000371.html
[2]https://github.com/freebsd/freebsd-src/blob/main/sys/ufs/ufs/inode.h#L75

FreeBSD Results (2)

- Inspecting tuples with relative support $0,9 \leq s_r < 1$
  - 11 tuples found
  - 9 false positives
    - No locks needed due to domain-specific knowledge[1]
    - Unguarded NULL-pointer checks
    - Locking pattern not covered by LockDoc[2]:
      - a Acquire *vnode lock* exclusively
      - b Use *vnode lock* in shared mode **+** *interlock*

[1]https://lists.freebsd.org/archives/freebsd-fs/2021-August/000371.html
[2]https://github.com/freebsd/freebsd-src/blob/main/sys/ufs/ufs/inode.h#L75

FreeBSD Results (2)

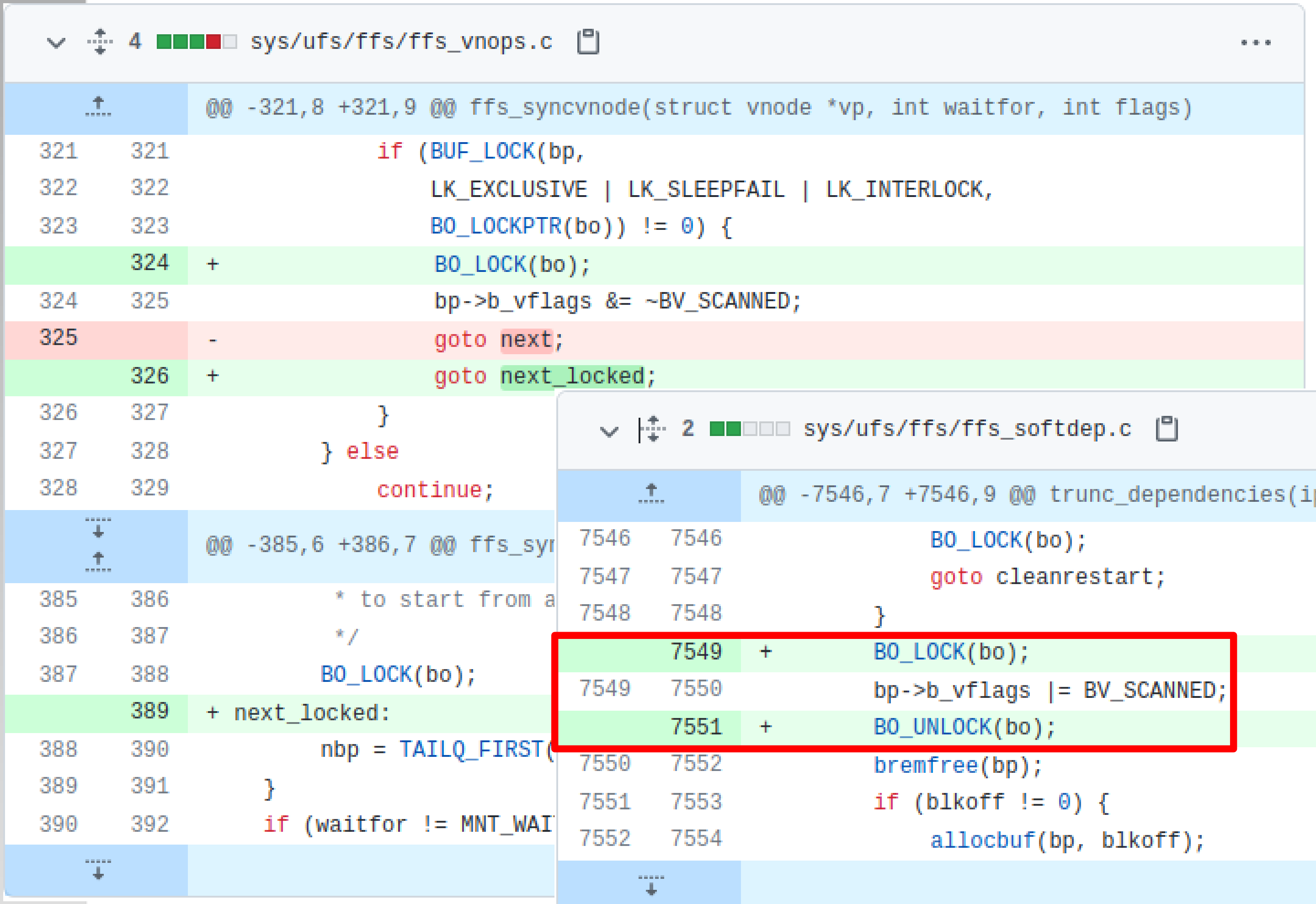
- Inspecting tuples with relative support $0{,}9 \leq s_r < 1$
  - 11 tuples found
  - 9 false positives
    - No locks needed due to domain-specific knowledge[1]
    - Unguarded NULL-pointer checks
    - Locking pattern not covered by LockDoc[2]:
      a. Acquire *vnode lock* exclusively
      b. Use *vnode lock* in shared mode + *interlock*
  - 2 real bugs
    - Relative support of 97.3 % and 96.2 %
    - Unguarded write to *buf.b_vflags* and read of *buf.b_blkno*

[1]https://lists.freebsd.org/archives/freebsd-fs/2021-August/000371.html
[2]https://github.com/freebsd/freebsd-src/blob/main/sys/ufs/ufs/inode.h#L75

technische universität dortmund

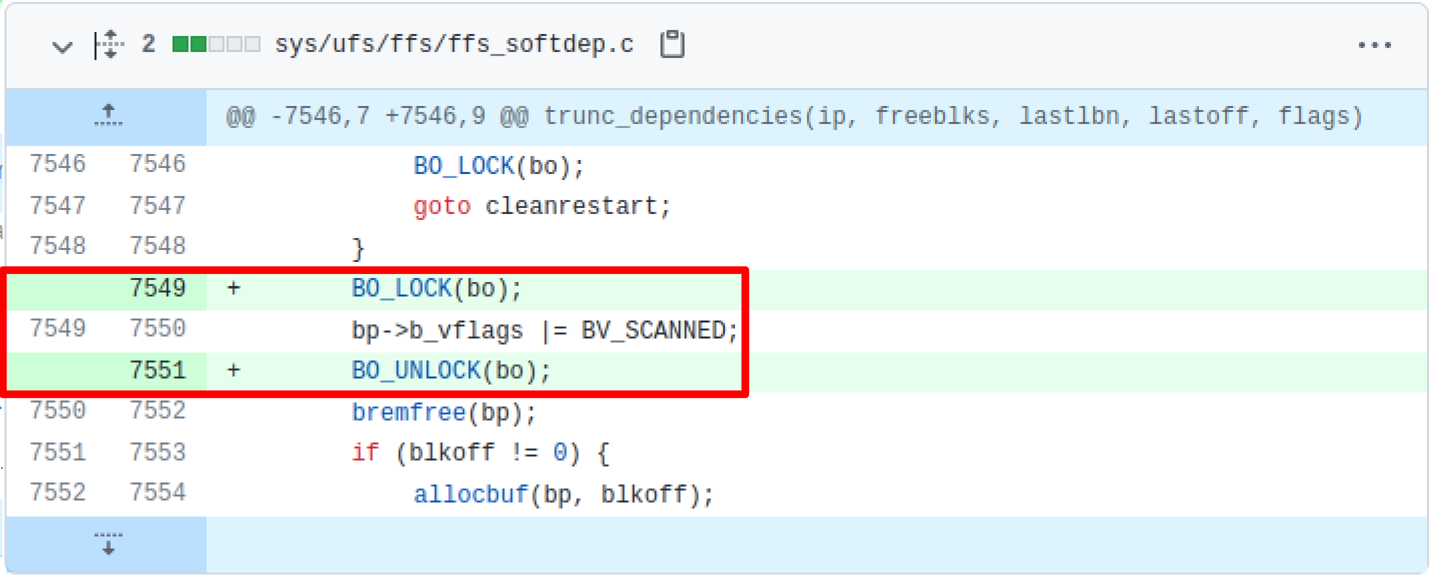
Locking Bug 1

- Unguarded write to *buf.b_vflags*



```
4 sys/ufs/ffs/ffs_vnops.c

@@ -321,8 +321,9 @@ ffs_syncvnode(struct vnode *vp, int waitfor, int flags)
                        if (BUF_LOCK(bp,
                            LK_EXCLUSIVE | LK_SLEEPFAIL | LK_INTERLOCK,
                            BO_LOCKPTR(bo)) != 0) {
+                               BO_LOCK(bo);
                                bp->b_vflags &= ~BV_SCANNED;
-                               goto next;
+                               goto next_locked;
                        }
                } else
                        continue;
@@ -385,6 +386,7 @@ ffs_sy
                         * to start from a
                         */
                        BO_LOCK(bo);
+ next_locked:
                        nbp = TAILQ_FIRST(
                }
                if (waitfor != MNT_WAI
```



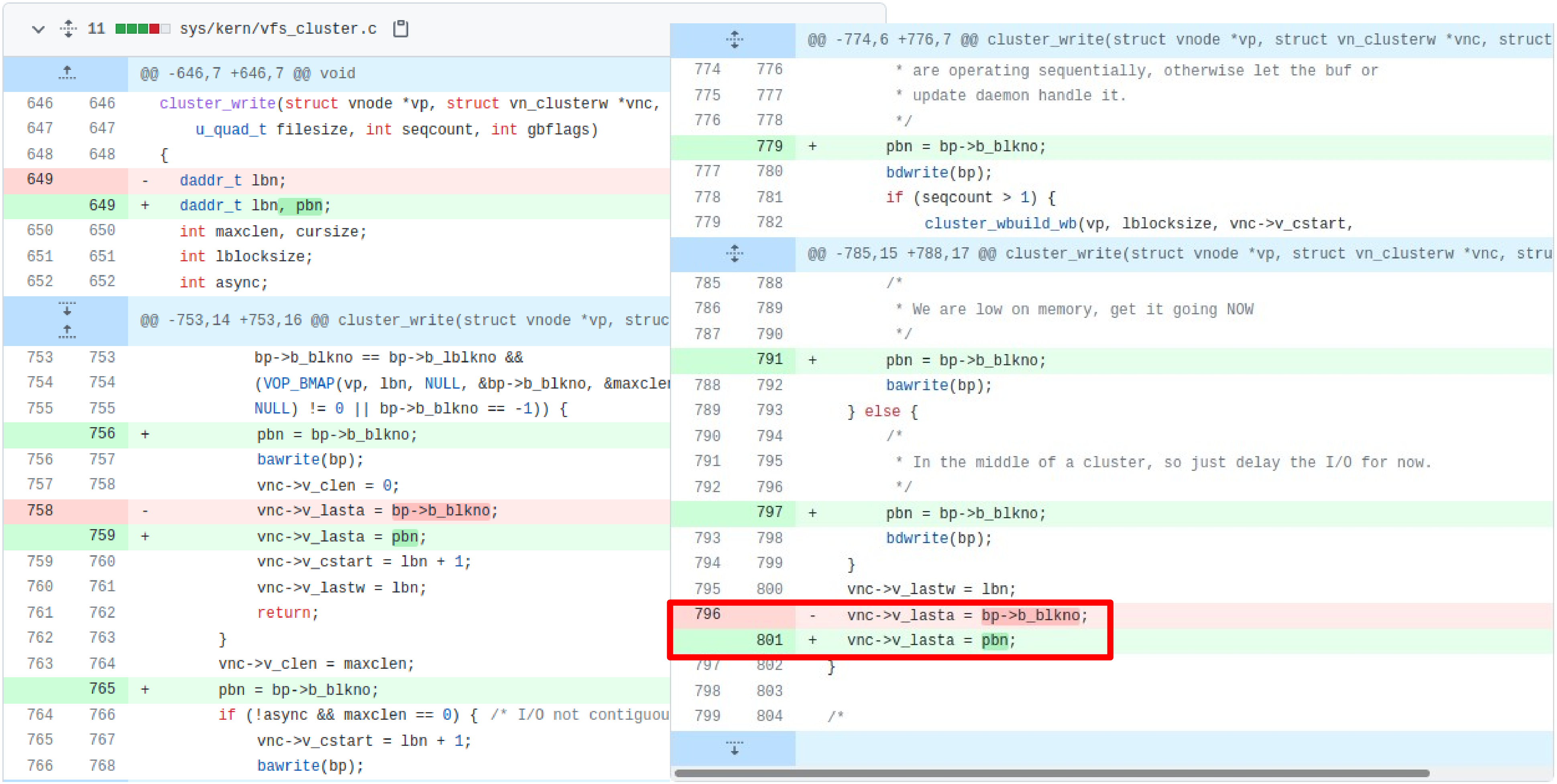
```
2 sys/ufs/ffs/ffs_softdep.c

@@ -7546,7 +7546,9 @@ trunc_dependencies(ip, freeblks, lastlbn, lastoff, flags)
                                BO_LOCK(bo);
                                goto cleanrestart;
                        }
+                       BO_LOCK(bo);
                        bp->b_vflags |= BV_SCANNED;
+                       BO_UNLOCK(bo);
                        bremfree(bp);
                        if (blkoff != 0) {
                                allocbuf(bp, blkoff);
```

https://github.com/freebsd/freebsd-src/commit/e3d675958539eee899d42438f5b46a26f3c64902

Locking Bug 2

- Unguarded read of *buf.b_blkno*



11 sys/kern/vfs_cluster.c

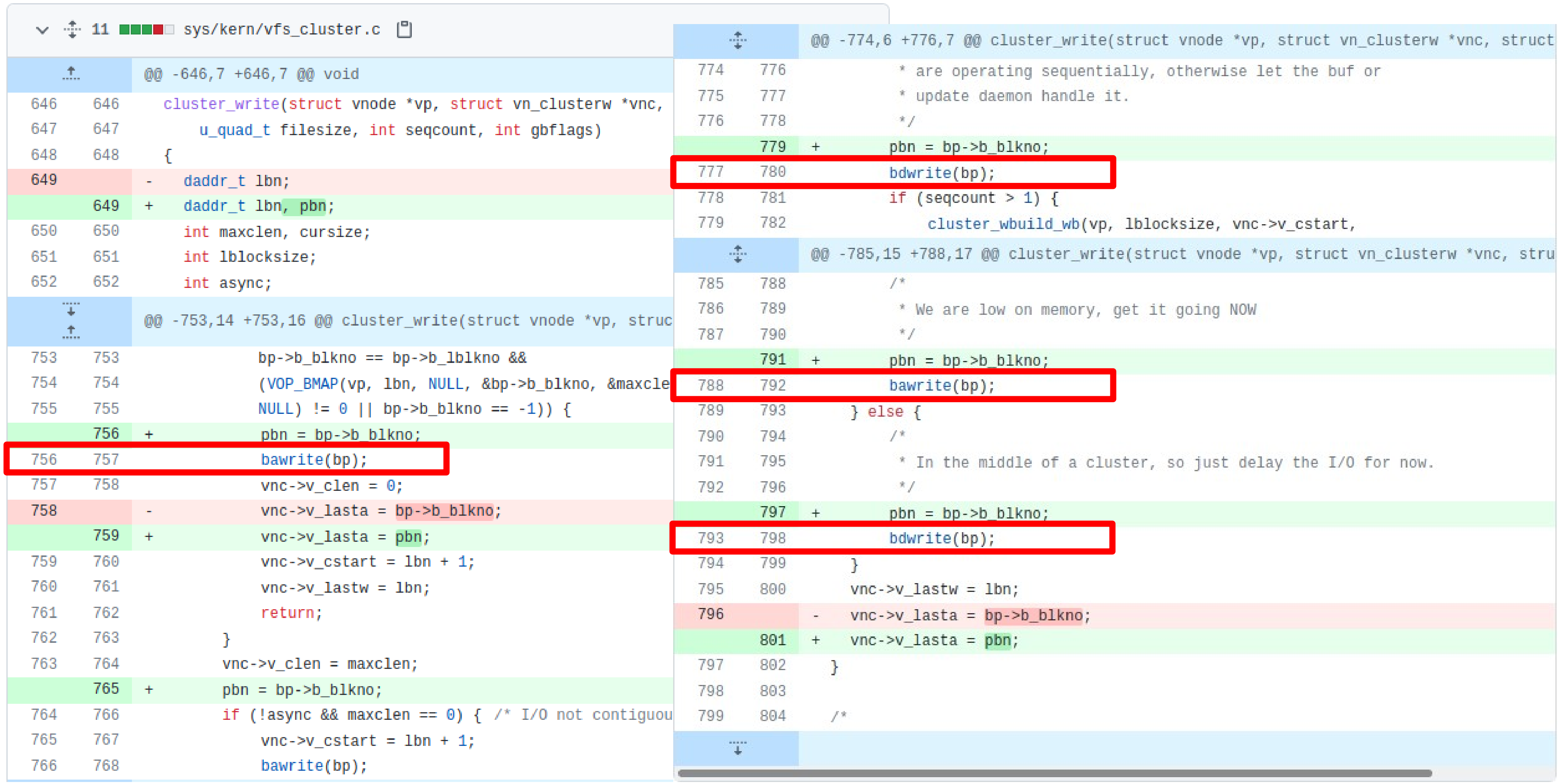
```
@@ -646,7 +646,7 @@ void
 cluster_write(struct vnode *vp, struct vn_clusterw *vnc,
     u_quad_t filesize, int seqcount, int gbflags)
 {
-	daddr_t lbn;
+	daddr_t lbn, pbn;
 	int maxclen, cursize;
 	int lblocksize;
 	int async;
@@ -753,14 +753,16 @@ cluster_write(struct vnode *vp, struc
 			    bp->b_blkno == bp->b_lblkno &&
 			    (VOP_BMAP(vp, lbn, NULL, &bp->b_blkno, &maxcle
 			    NULL) != 0 || bp->b_blkno == -1)) {
+				pbn = bp->b_blkno;
 				bawrite(bp);
 				vnc->v_clen = 0;
-				vnc->v_lasta = bp->b_blkno;
+				vnc->v_lasta = pbn;
 				vnc->v_cstart = lbn + 1;
 				vnc->v_lastw = lbn;
 				return;
 			}
 			vnc->v_clen = maxclen;
+			pbn = bp->b_blkno;
 			if (!async && maxclen == 0) {	/* I/O not contiguou
 				vnc->v_cstart = lbn + 1;
 				bawrite(bp);
```

```
@@ -774,6 +776,7 @@ cluster_write(struct vnode *vp, struct vn_clusterw *vnc, struct
 			 * are operating sequentially, otherwise let the buf or
 			 * update daemon handle it.
 			 */
+			pbn = bp->b_blkno;
 			bdwrite(bp);
 			if (seqcount > 1) {
 				cluster_wbuild_wb(vp, lblocksize, vnc->v_cstart,
@@ -785,15 +788,17 @@ cluster_write(struct vnode *vp, struct vn_clusterw *vnc, stru
 			/*
 			 * We are low on memory, get it going NOW
 			 */
+			pbn = bp->b_blkno;
 			bawrite(bp);
 		} else {
 			/*
 			 * In the middle of a cluster, so just delay the I/O for now.
 			 */
+			pbn = bp->b_blkno;
 			bdwrite(bp);
 		}
 		vnc->v_lastw = lbn;
-		vnc->v_lasta = bp->b_blkno;
+		vnc->v_lasta = pbn;
 	}
 
 	/*
```

https://github.com/freebsd/freebsd-src/commit/5cc82c5

Locking Bug 2

- Unguarded read of *buf.b_blkno*



```
11 sys/kern/vfs_cluster.c

@@ -646,7 +646,7 @@ void
 cluster_write(struct vnode *vp, struct vn_clusterw *vnc,
     u_quad_t filesize, int seqcount, int gbflags)
 {
-	daddr_t lbn;
+	daddr_t lbn, pbn;
 	int maxclen, cursize;
 	int lblocksize;
 	int async;
@@ -753,14 +753,16 @@ cluster_write(struct vnode *vp, struc
 			bp->b_blkno == bp->b_lblkno &&
 			(VOP_BMAP(vp, lbn, NULL, &bp->b_blkno, &maxcle
 			NULL) != 0 || bp->b_blkno == -1)) {
+				pbn = bp->b_blkno;
 				bawrite(bp);
 				vnc->v_clen = 0;
-				vnc->v_lasta = bp->b_blkno;
+				vnc->v_lasta = pbn;
 				vnc->v_cstart = lbn + 1;
 				vnc->v_lastw = lbn;
 				return;
 			}
 			vnc->v_clen = maxclen;
+			pbn = bp->b_blkno;
 			if (!async && maxclen == 0) {	/* I/O not contiguou
 				vnc->v_cstart = lbn + 1;
 				bawrite(bp);
```

```
@@ -774,6 +776,7 @@ cluster_write(struct vnode *vp, struct vn_clusterw *vnc, struct
 			 * are operating sequentially, otherwise let the buf or
 			 * update daemon handle it.
 			 */
+			pbn = bp->b_blkno;
 			bdwrite(bp);
 			if (seqcount > 1) {
 				cluster_wbuild_wb(vp, lblocksize, vnc->v_cstart,
@@ -785,15 +788,17 @@ cluster_write(struct vnode *vp, struct vn_clusterw *vnc, stru
 			/*
 			 * We are low on memory, get it going NOW
 			 */
+			pbn = bp->b_blkno;
 			bawrite(bp);
 		} else {
 			/*
 			 * In the middle of a cluster, so just delay the I/O for now.
 			 */
+			pbn = bp->b_blkno;
 			bdwrite(bp);
 		}
 		vnc->v_lastw = lbn;
-		vnc->v_lasta = bp->b_blkno;
+		vnc->v_lasta = pbn;
 	}
 
 	/*
```

https://github.com/freebsd/freebsd-src/commit/5cc82c5

FreeBSD Results (2)

- Inspecting tuples with relative support $0{,}9 \leq s_r < 1$
  - 11 tuples found
  - 9 false positives
    - No locks needed due to domain-specific knowledge
    - Unguarded NULL-pointer checks
    - Pattern not covered by LockDoc:
      - a Acquire *vnode lock* exclusively
      - b Use *vnode lock* in shared mode + *interlock*
  - 2 real bugs
    - Relative support of 97.3 % and 96.2 %
    - Unguarded write to *buf.b_vflags* and read of *buf.b_blkno*
- *Taking samples from tuples with relative support* $s_r < 0.9$
  - *buf.b_qindex* and *buf.b_subqueue are "Protected by the buf queue lock"*[1]

[1] cf. sys/sys/buf.h, line 96

technische universität
dortmund

FreeBSD Results (3)

- *buf.b_qindex* and *buf.b_subqueue are*
  “Protected by the buf queue lock“[1]

The “buf queue lock”:

```
struct bufqueue {
      struct mtx_padalign   bq_lock;
      // [...]
};
```

Multiple buf queues exist:

```
struct bufdomain {
      struct bufqueue        bd_subq[MAXCPU + 1]; /* [...] */
      struct bufqueue        bd_dirtyq;
      struct bufqueue        *bd_cleanq;
      struct mtx_padalign bd_run_lock;
      // [...]
};
```

- Multiple locks exist:
  - ***bq_subq.bq_lock***
  - ***bq_dirtyq.bq_lock***
- Accesses are split across them

[1] cf. sys/sys/buf.h, line 96

Summary

- 72.41 % of all observed fields adhere to locking documentation
- Using sound locking documentation to search for bugs
  - Found limitations of LockDoc
  - Found **2 locking bugs**
- Outlook
  - Integrate lock classes, e.g., *bq_subq.bq_lock* ↔ *bq_dirtyq.bq_lock*
  - Further investigate rules with rel. support < 90 %

Icon Licenses: CC-BY-SA 4.0 / CC0 1.0
Emoji License: CC-BY-SA 4.0 by Emoji One
FreeBSD logo – A Trademark of FreeBSD Foundation
Tux by Larry Ewing lewing@isc.tamu.edu

References

[1] Robert Love. 2010. *Linux Kernel Development (3rd ed.)*.

[2] Marshall Kirk McKusick, George V. Neville-Neil, and Robert N. M. Watson. 2014. *The Design and Implementation of the FreeBSD Operating System*.

[3]Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys‘19.

[4]https://github.com/linux-test-project/ltp