SYNACKTIV

Ubuntu Shiftfs: Unbalanced Unlock Exploitation Attempt

CVE-2023-2612

Grehack 2023

Agenda

- Introduction
- Code review and bug discovery
- Exploitation
- Conclusion

Presentation

- **Jean-Baptiste Cayrou**
  - Security researcher @Synacktiv
  - In the Reverse Engineering team
- **Synacktiv**
  - Offensive security company
  - Based in France
  - ~170 Ninjas
  - We are hiring!

@jbcayrou

Introduction

- **Targeting Ubuntu for Pwn2Own Vancouver (May 18, 2022)**
- **Need to find and exploit a kernel vulnerability to gain root access**
- **Motivations**
  - Learn more about Linux kernel internals
  - Learn new techniques to exploit kernel vulnerabilities
  - $40,000 bounty

Kernel code review and vulnerability research

Code Review - Attack surface

- **Kernel source code is huge, where to start ?**
- **Previous success from my colleague *Vincent Dehors***
  - Found and exploited an Ubuntu vulnerability for the contest
  - His work is described in his blog post
- **TLDR: Look for vulnerabilities in uncommon surfaces that are less audited**
  - By default Ubuntu allows users to create namespaces

```
$ sudo sysctl kernel.unprivileged_userns_clone
kernel.unprivileged_userns_clone = 1
```

Code Review - Namespace introduction

- **Namespace is a feature that provides process isolation**
- **Used to create a separate set of resources**
- **Useful for creating containers (such as docker, LXC, etc.)**
- **Types of namespaces**
  - mount - Isolates filesystem mount points → Focus on this one
  - process ID
  - network
  - IPC
  - ...

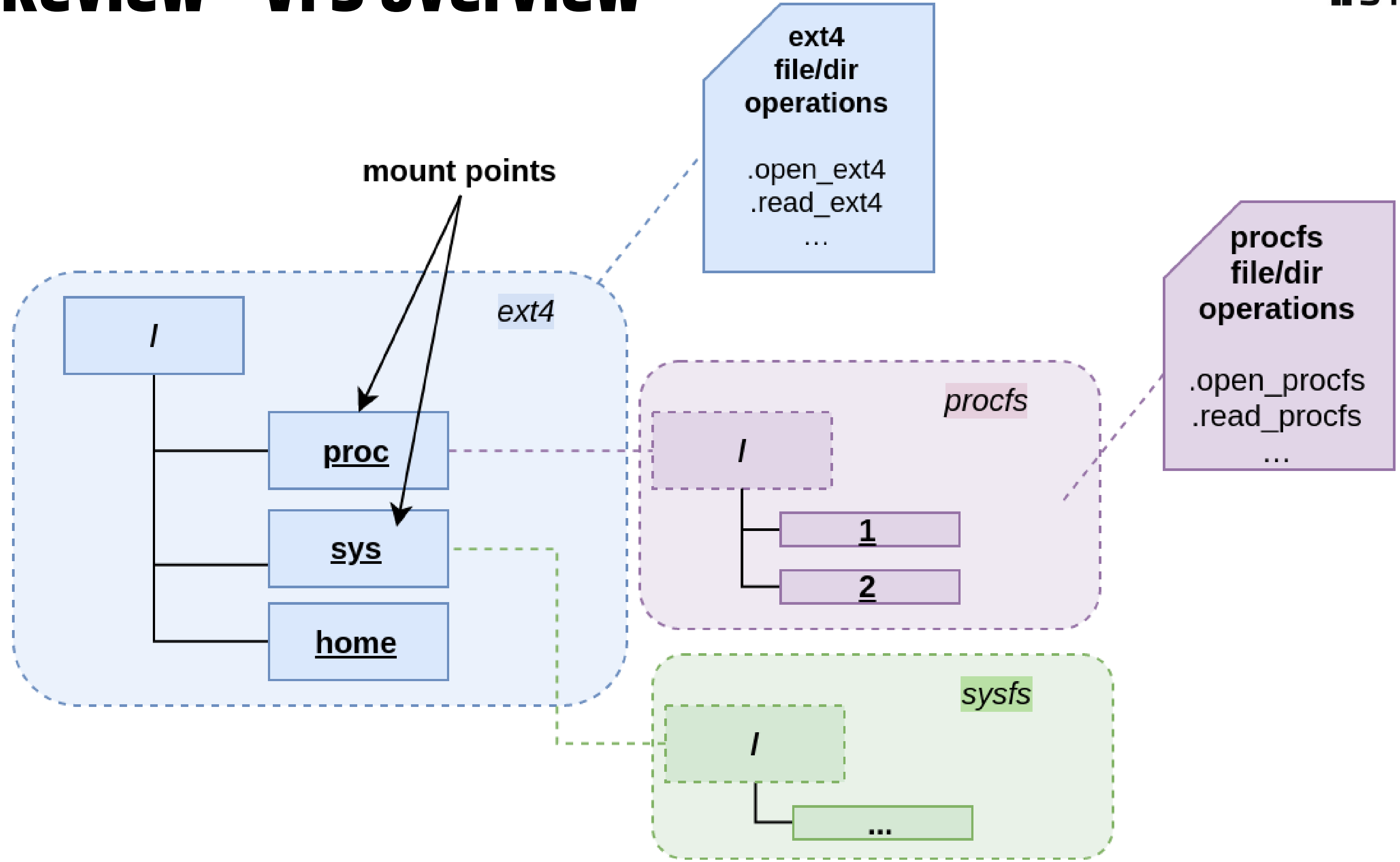
Code Review - mount and file systems

- **Filesystems that have the flag *FS_USERNS_MOUNT* can be set up by a unprivileged user**

```
static struct file_system_type shiftfs_type = {
     .owner        = THIS_MODULE,
     .name         = "shiftfs",
     .mount        = shiftfs_mount,
     .kill_sb= kill_anon_super,
     .fs_flags     = FS_USERNS_MOUNT,
};
```

- **VFS surfaces:**
  - android/binderfs
  - mqueue
  - shmem
  - sysfs
  - ramfs (tmpfs)
  - overlayfs
  - proc
  - aufs
  - fuse
  - shiftfs (specific to Ubuntu)
  - devpts
  - cgroup

Code Review - VFS overview



Code Review - Attack surface

- **File manipulations**
  - open, read, write, fnctl ...
  - Race condition on concurrent access
  - Logical bugs
- **Mounting syscalls and options**
  - mount, fsopen, fspick, fsconfig

```
mount [-fnrsvw] [-t fstype] [-o options] device mountpoint
```

Code Review - Read the doc !

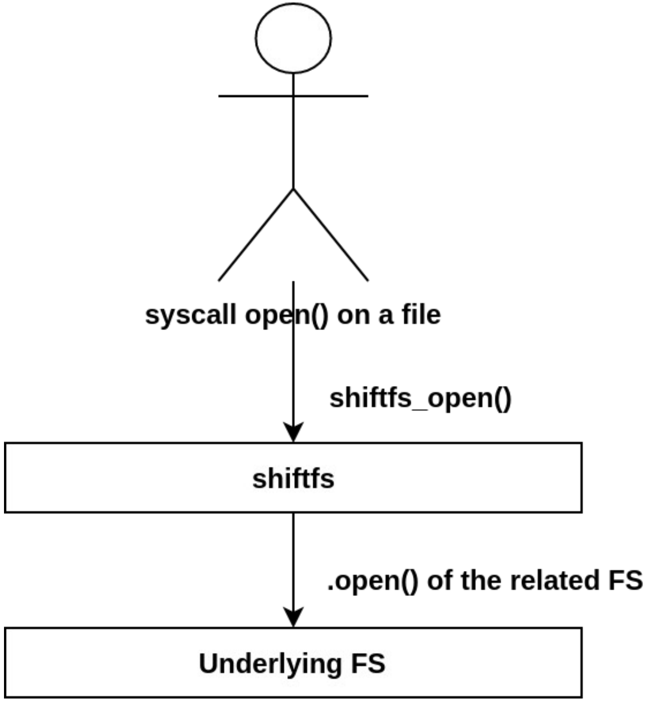
- **Read Kernel VFS documentation**
  - Learn how this kernel subsystem works
- **Study past CVEs that affected kernel Filesystems**
  - Study errors that should not be made
- **Read blog posts about kernel exploitation**

Code Review - Let's go

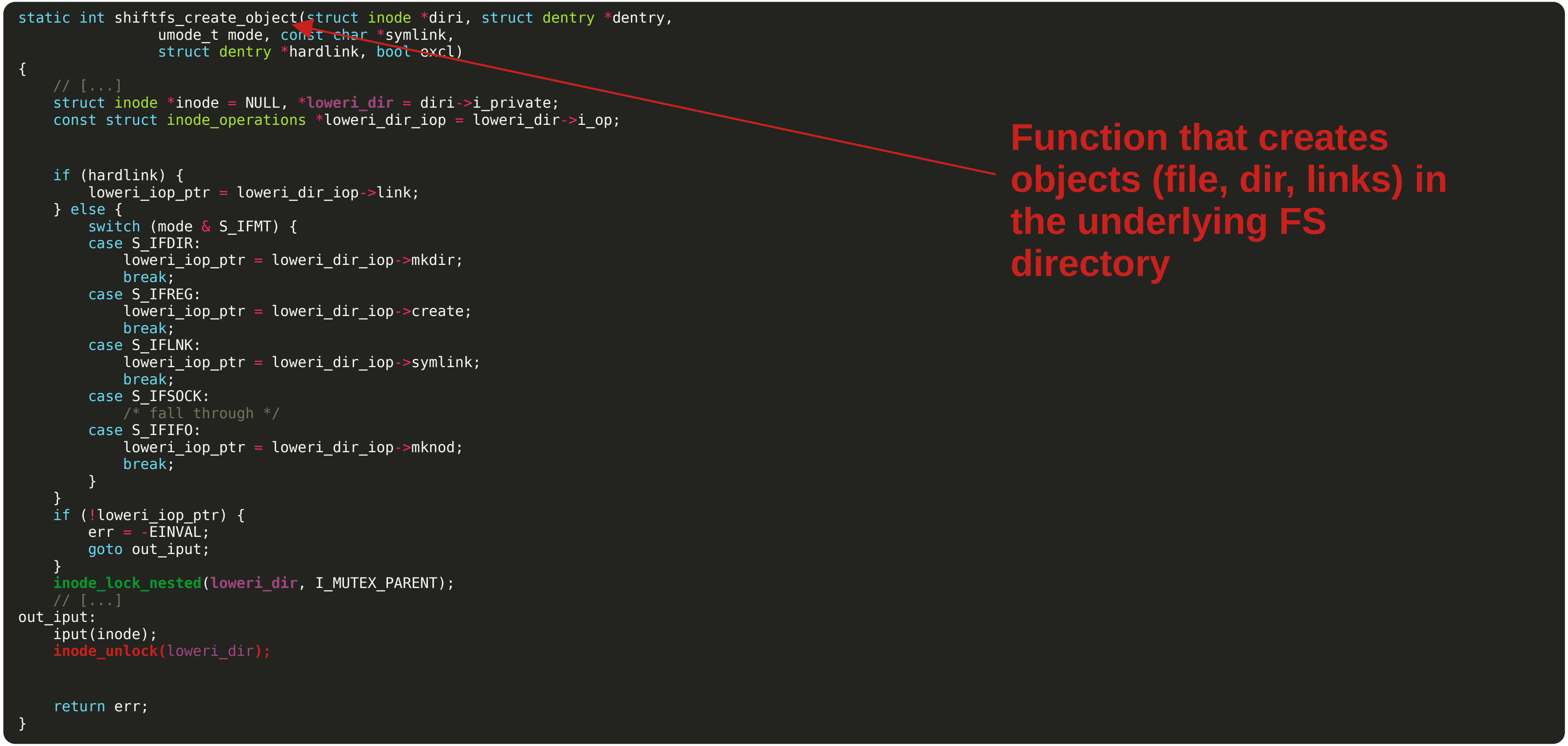
- **Review accessible filesystems one by one**
  - Skip *shiftfs* because my colleague already found things in it!
- **About 3 weeks (not in full time)**
  - Still nothing ...
- **Start looking at shiftfs implementation**
  - BINGO! There is a bug!

Code Review - shiftfs overview

- **This filesystem is a passthrough used to change (shift) the user unix permissions on file access or modification**



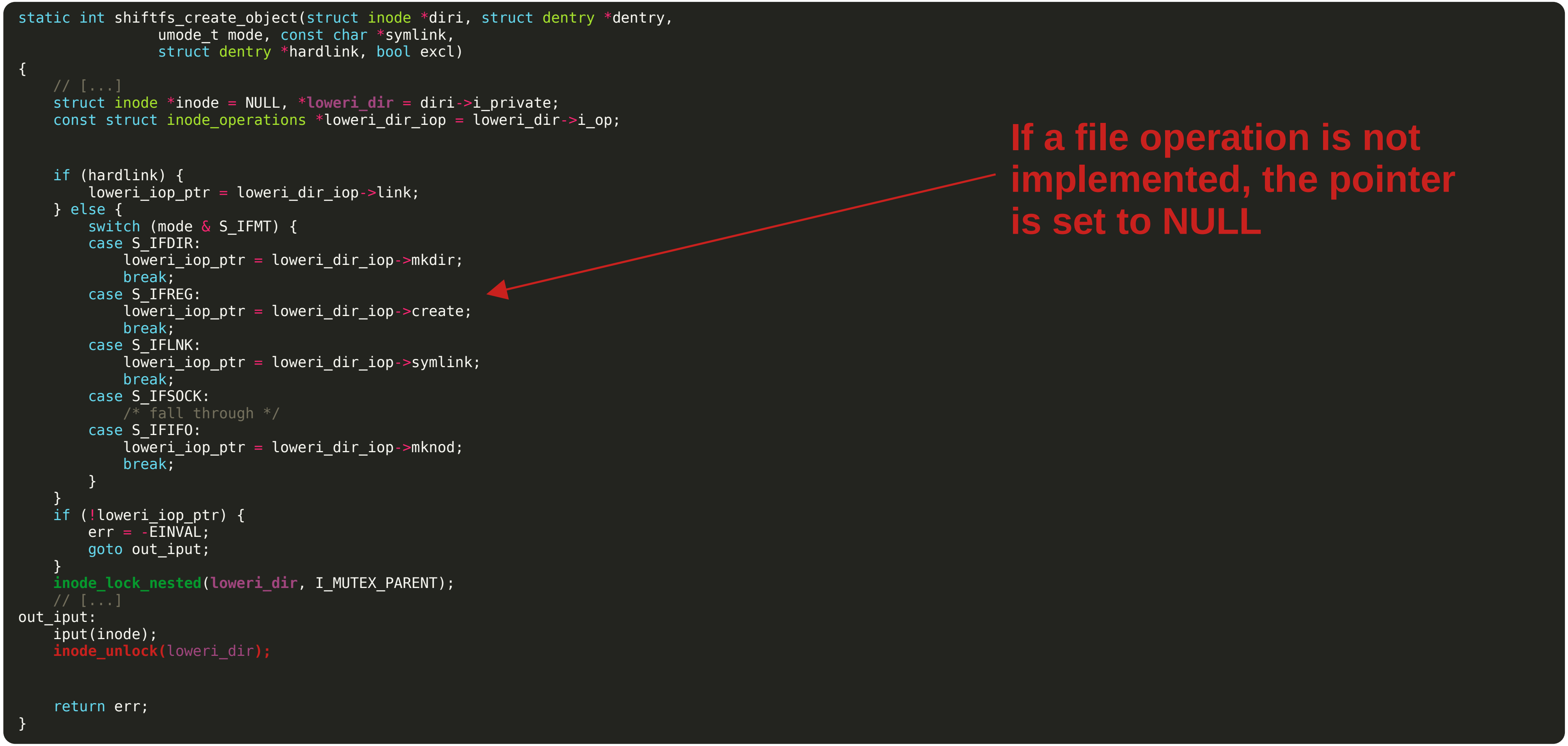
Code Review - The lock shiftfs bug



```
static int shiftfs_create_object(struct inode *diri, struct dentry *dentry,
                umode_t mode, const char *symlink,
                struct dentry *hardlink, bool excl)
{
    // [...]
    struct inode *inode = NULL, *loweri_dir = diri->i_private;
    const struct inode_operations *loweri_dir_iop = loweri_dir->i_op;


    if (hardlink) {
        loweri_iop_ptr = loweri_dir_iop->link;
    } else {
        switch (mode & S_IFMT) {
        case S_IFDIR:
            loweri_iop_ptr = loweri_dir_iop->mkdir;
            break;
        case S_IFREG:
            loweri_iop_ptr = loweri_dir_iop->create;
            break;
        case S_IFLNK:
            loweri_iop_ptr = loweri_dir_iop->symlink;
            break;
        case S_IFSOCK:
            /* fall through */
        case S_IFIFO:
            loweri_iop_ptr = loweri_dir_iop->mknod;
            break;
        }
    }
    if (!loweri_iop_ptr) {
        err = -EINVAL;
        goto out_iput;
    }
    inode_lock_nested(loweri_dir, I_MUTEX_PARENT);
    // [...]
out_iput:
    iput(inode);
    inode_unlock(loweri_dir);


    return err;
}
```

Function that creates objects (file, dir, links) in the underlying FS directory

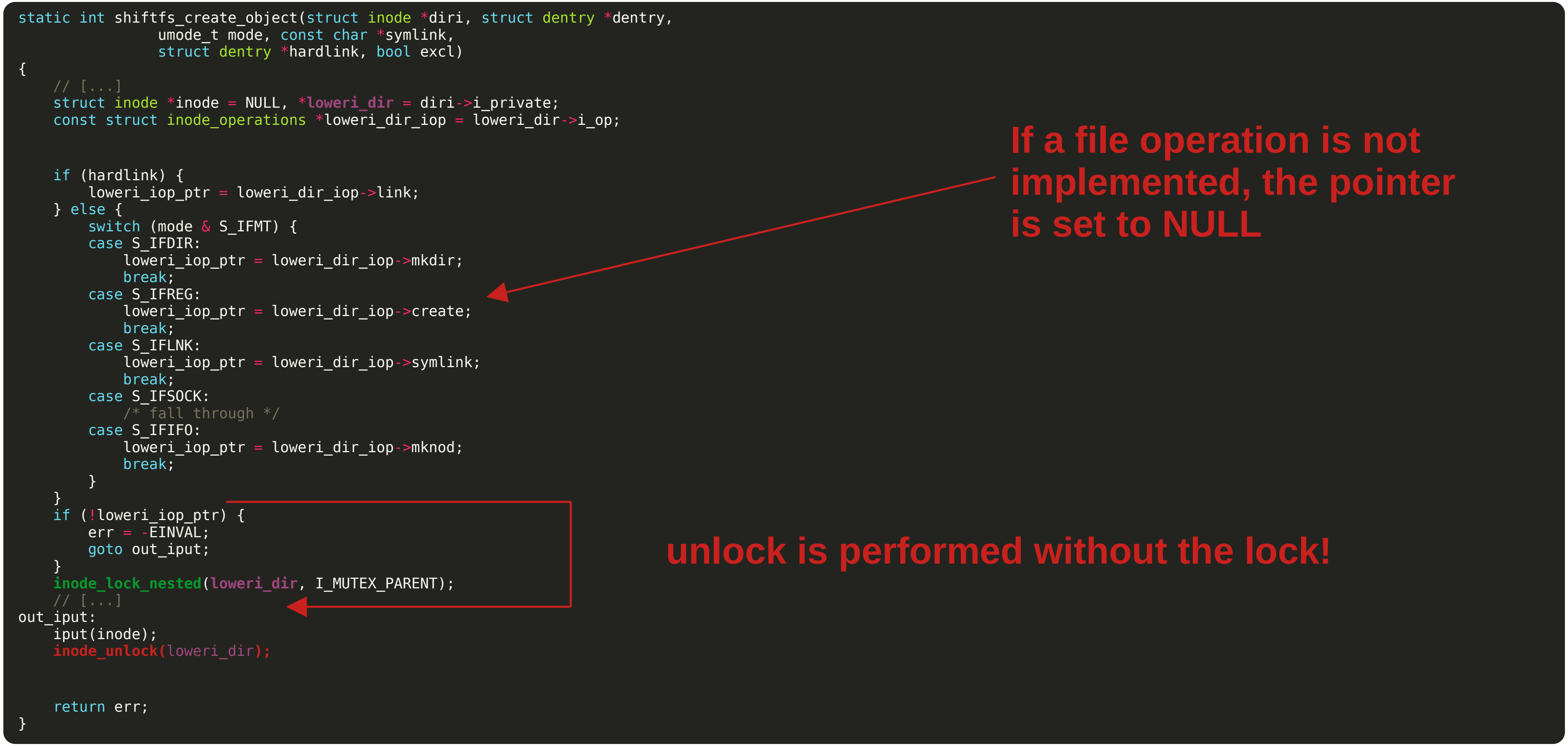
Code Review - The lock shiftfs bug



```
static int shiftfs_create_object(struct inode *diri, struct dentry *dentry,
                umode_t mode, const char *symlink,
                struct dentry *hardlink, bool excl)
{
    // [...]
    struct inode *inode = NULL, *loweri_dir = diri->i_private;
    const struct inode_operations *loweri_dir_iop = loweri_dir->i_op;


    if (hardlink) {
        loweri_iop_ptr = loweri_dir_iop->link;
    } else {
        switch (mode & S_IFMT) {
        case S_IFDIR:
            loweri_iop_ptr = loweri_dir_iop->mkdir;
            break;
        case S_IFREG:
            loweri_iop_ptr = loweri_dir_iop->create;
            break;
        case S_IFLNK:
            loweri_iop_ptr = loweri_dir_iop->symlink;
            break;
        case S_IFSOCK:
            /* fall through */
        case S_IFIFO:
            loweri_iop_ptr = loweri_dir_iop->mknod;
            break;
        }
    }
    if (!loweri_iop_ptr) {
        err = -EINVAL;
        goto out_iput;
    }
    inode_lock_nested(loweri_dir, I_MUTEX_PARENT);
    // [...]
out_iput:
    iput(inode);
    inode_unlock(loweri_dir);


    return err;
}
```

If a file operation is not implemented, the pointer is set to NULL

Code Review - The lock shiftfs bug



```
static int shiftfs_create_object(struct inode *diri, struct dentry *dentry,
                umode_t mode, const char *symlink,
                struct dentry *hardlink, bool excl)
{
    // [...]
    struct inode *inode = NULL, *loweri_dir = diri->i_private;
    const struct inode_operations *loweri_dir_iop = loweri_dir->i_op;


    if (hardlink) {
        loweri_iop_ptr = loweri_dir_iop->link;
    } else {
        switch (mode & S_IFMT) {
        case S_IFDIR:
            loweri_iop_ptr = loweri_dir_iop->mkdir;
            break;
        case S_IFREG:
            loweri_iop_ptr = loweri_dir_iop->create;
            break;
        case S_IFLNK:
            loweri_iop_ptr = loweri_dir_iop->symlink;
            break;
        case S_IFSOCK:
            /* fall through */
        case S_IFIFO:
            loweri_iop_ptr = loweri_dir_iop->mknod;
            break;
        }
    }
    if (!loweri_iop_ptr) {
        err = -EINVAL;
        goto out_iput;
    }
    inode_lock_nested(loweri_dir, I_MUTEX_PARENT);
    // [...]
out_iput:
    iput(inode);
    inode_unlock(loweri_dir);


    return err;
}
```

If a file operation is not implemented, the pointer is set to NULL

unlock is performed without the lock!

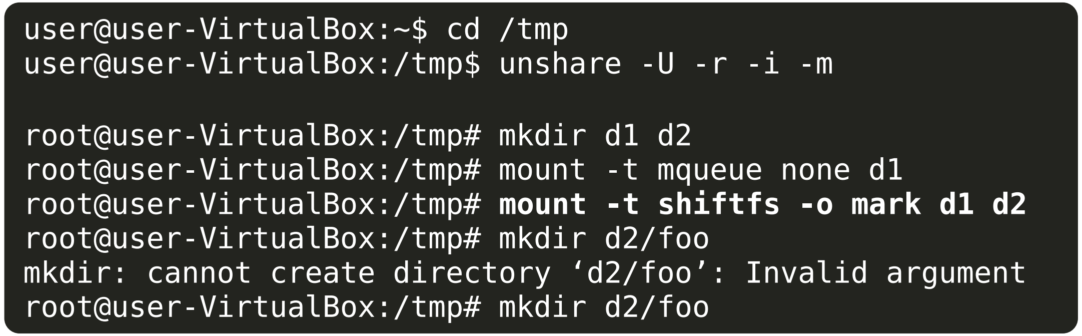
Code Review - Trigger the bug

- **Find a filesystem (*FS_USERNS_MOUNT*) that does not implement *mkdir*, *create*, *symlink*, *link* or *mknod* in its *inode_operations* structure**
  - *mqueue* is a good candidate

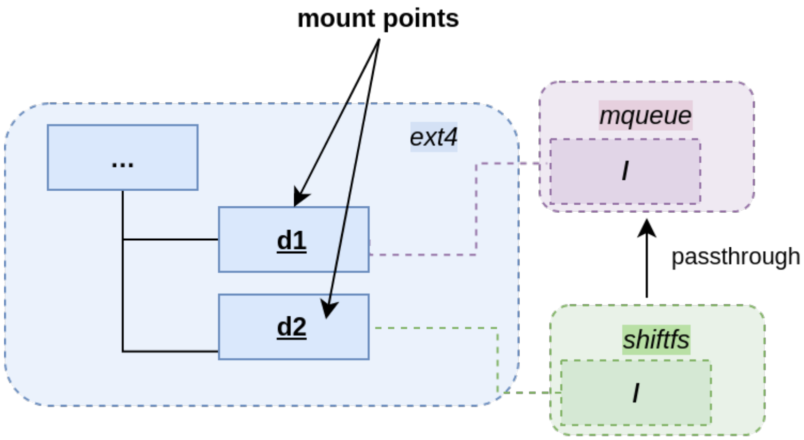
```
// Extract from ipc/mqueue.c
static const struct inode_operations mqueue_dir_inode_operations = {
    .lookup = simple_lookup,
    .create = mqueue_create,
    .unlink = mqueue_unlink,
};
```

Code Review - Trigger the bug

- **Trigger the bug**

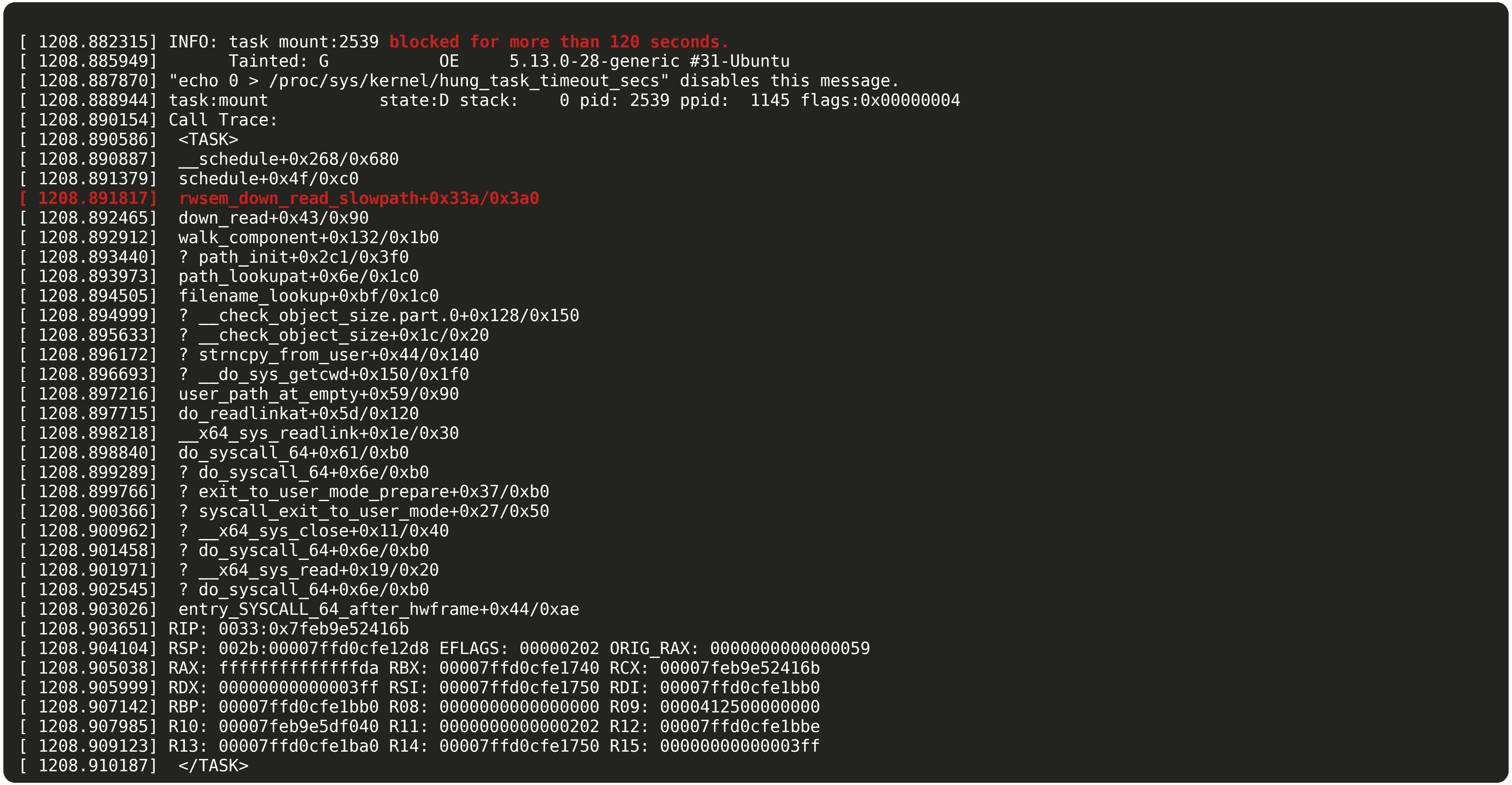


```
user@user-VirtualBox:~$ cd /tmp
user@user-VirtualBox:/tmp$ unshare -U -r -i -m

root@user-VirtualBox:/tmp# mkdir d1 d2
root@user-VirtualBox:/tmp# mount -t mqueue none d1
root@user-VirtualBox:/tmp# mount -t shiftfs -o mark d1 d2
root@user-VirtualBox:/tmp# mkdir d2/foo
mkdir: cannot create directory ‘d2/foo’: Invalid argument
root@user-VirtualBox:/tmp# mkdir d2/foo
```



- **The last “mkdir *d2/foo*” *is now blocked…***
- **After several seconds**

Code Review - Trigger the bug



```
[ 1208.882315] INFO: task mount:2539 blocked for more than 120 seconds.
[ 1208.885949]       Tainted: G           OE     5.13.0-28-generic #31-Ubuntu
[ 1208.887870] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
[ 1208.888944] task:mount           state:D stack:    0 pid: 2539 ppid:  1145 flags:0x00000004
[ 1208.890154] Call Trace:
[ 1208.890586]  <TASK>
[ 1208.890887]  __schedule+0x268/0x680
[ 1208.891379]  schedule+0x4f/0xc0
[ 1208.891817]  rwsem_down_read_slowpath+0x33a/0x3a0
[ 1208.892465]  down_read+0x43/0x90
[ 1208.892912]  walk_component+0x132/0x1b0
[ 1208.893440]  ? path_init+0x2c1/0x3f0
[ 1208.893973]  path_lookupat+0x6e/0x1c0
[ 1208.894505]  filename_lookup+0xbf/0x1c0
[ 1208.894999]  ? __check_object_size.part.0+0x128/0x150
[ 1208.895633]  ? __check_object_size+0x1c/0x20
[ 1208.896172]  ? strncpy_from_user+0x44/0x140
[ 1208.896693]  ? __do_sys_getcwd+0x150/0x1f0
[ 1208.897216]  user_path_at_empty+0x59/0x90
[ 1208.897715]  do_readlinkat+0x5d/0x120
[ 1208.898218]  __x64_sys_readlink+0x1e/0x30
[ 1208.898840]  do_syscall_64+0x61/0xb0
[ 1208.899289]  ? do_syscall_64+0x6e/0xb0
[ 1208.899766]  ? exit_to_user_mode_prepare+0x37/0xb0
[ 1208.900366]  ? syscall_exit_to_user_mode+0x27/0x50
[ 1208.900962]  ? __x64_sys_close+0x11/0x40
[ 1208.901458]  ? do_syscall_64+0x6e/0xb0
[ 1208.901971]  ? __x64_sys_read+0x19/0x20
[ 1208.902545]  ? do_syscall_64+0x6e/0xb0
[ 1208.903026]  entry_SYSCALL_64_after_hwframe+0x44/0xae
[ 1208.903651] RIP: 0033:0x7feb9e52416b
[ 1208.904104] RSP: 002b:00007ffd0cfe12d8 EFLAGS: 00000202 ORIG_RAX: 0000000000000059
[ 1208.905038] RAX: ffffffffffffffda RBX: 00007ffd0cfe1740 RCX: 00007feb9e52416b
[ 1208.905999] RDX: 00000000000003ff RSI: 00007ffd0cfe1750 RDI: 00007ffd0cfe1bb0
[ 1208.907142] RBP: 00007ffd0cfe1bb0 R08: 0000000000000000 R09: 0000412500000000
[ 1208.907985] R10: 00007feb9e5df040 R11: 0000000000000202 R12: 00007ffd0cfe1bbe
[ 1208.909123] R13: 00007ffd0cfe1ba0 R14: 00007ffd0cfe1750 R15: 00000000000003ff
[ 1208.910187]  </TASK>
```

Exploitation

(How to get root with this bug?)

Exploitation - The objective

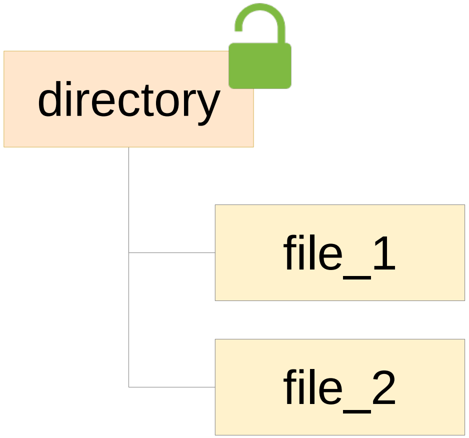
- **Perform a Local Privilege Escalation (LPE) and get root**
  - Need to modify our process permissions to change the UID to 0 (root user)
- **We do not need kernel code execution**
  - Having kernel read and write primitives is enough
  - We also need a kernel pointer leak
    - To bypass the KASLR
    - To locate the data related to our process in the kernel memory

Exploitation - Side effect of the bug ?

- **How to turn this locking bug into something useful ?**
- **The bug unlocks a directory lock**
  - What does it protect?
  - What could happen if such a lock is wrongfully unlocked?

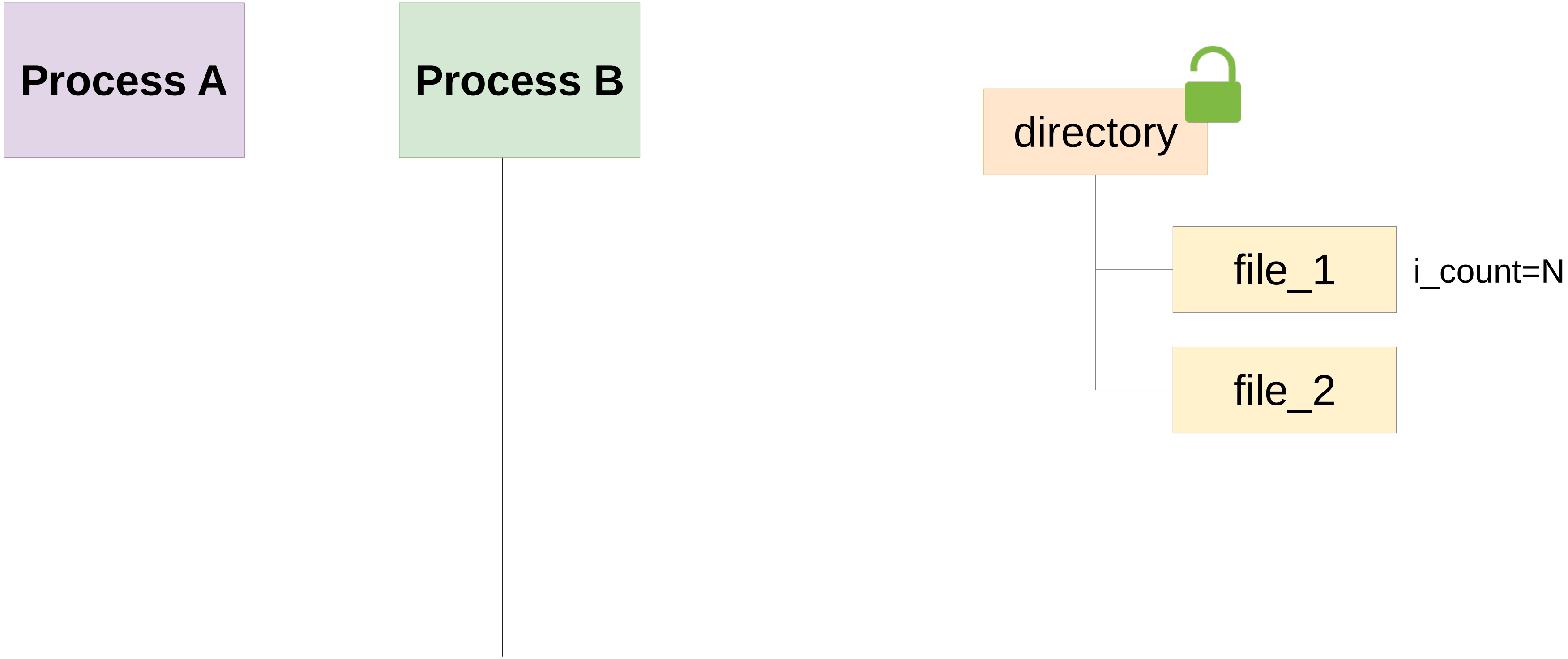
Exploitation - Example lock usage

- **When the content of the directory is modified the lock is taken**
  - Create a file, a folder, a link
  - Remove a file
- **This prevents concurrent access and race conditions during directory modifications**



Example of a mqueue FS

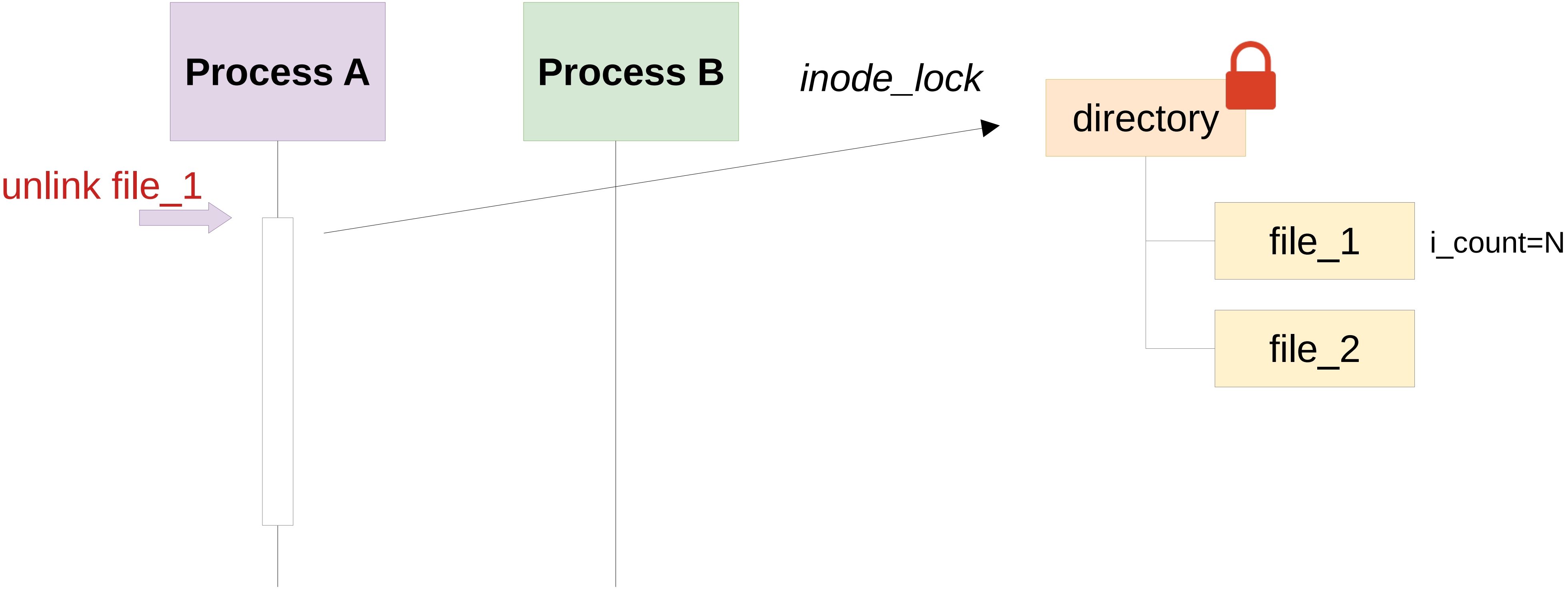
Exploitation - Example lock usage: removing a file



Note: i_count is the inode usage count. When it hits zero, it is freed

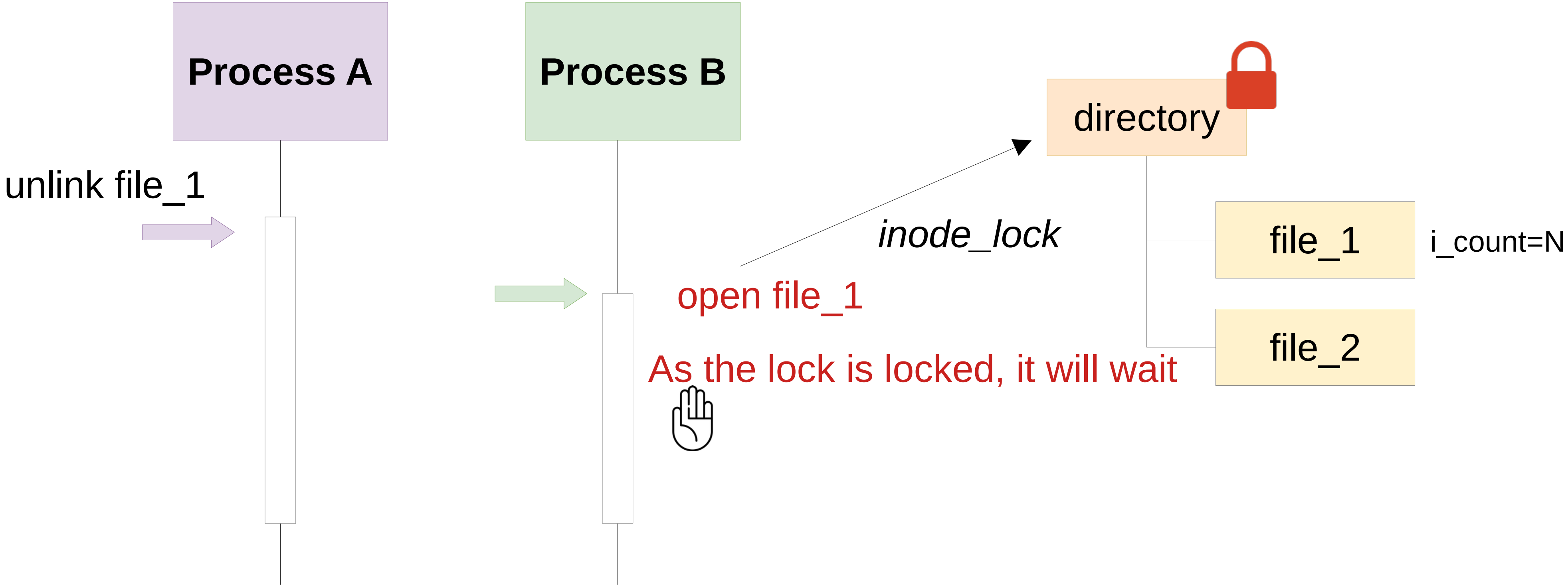
Exploitation - Example lock usage: removing a file

- **Process A starts to remove a file, the directory inode is locked**



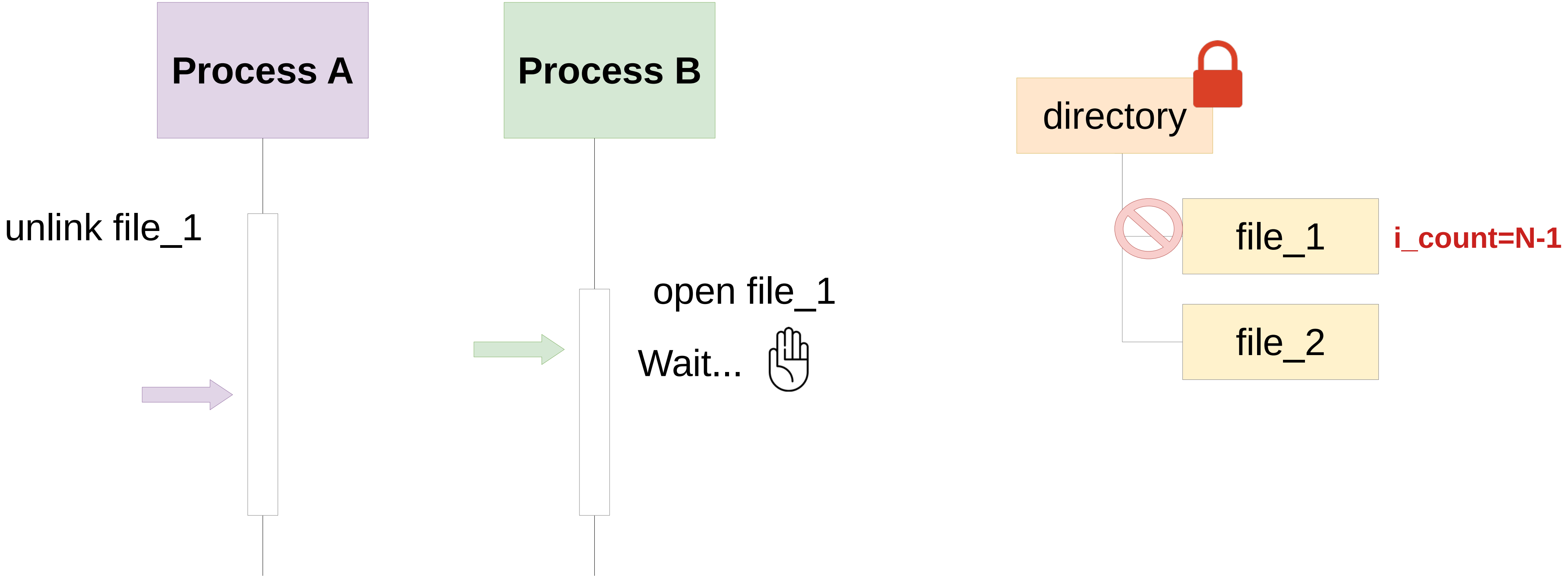
Exploitation - Example lock usage: removing a file

- At the same time, Process B wants to open it



Exploitation - Example lock usage: removing a file

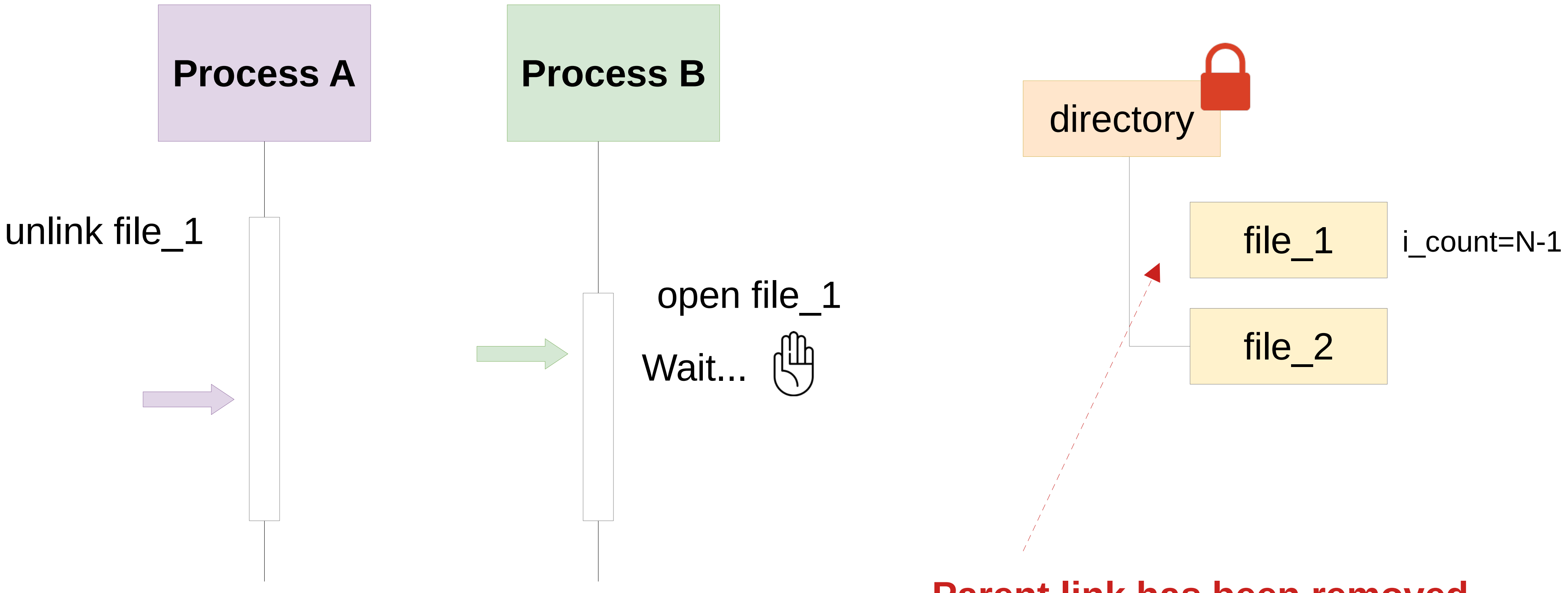
- **Process A removes the link and decrements the usage counter**



Exploitation - Example lock usage: removing a file

SYNACKTIV

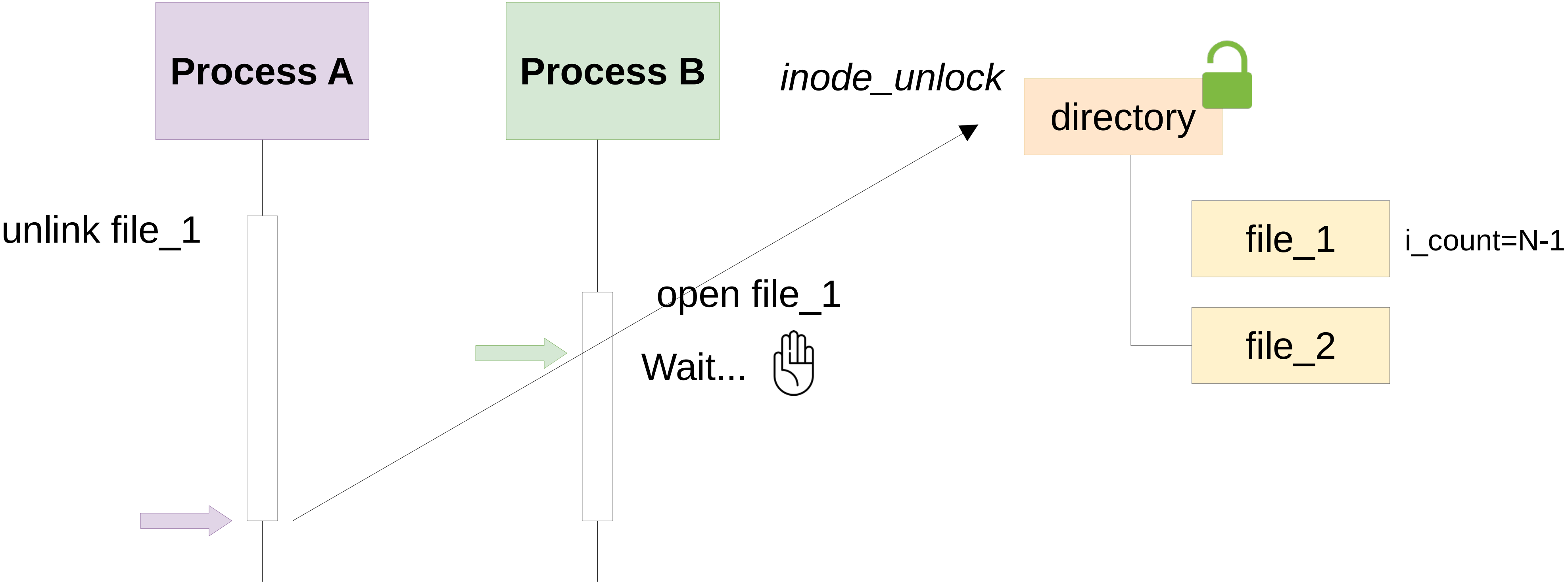
- Process A continues the unlink ...



Parent link has been removed

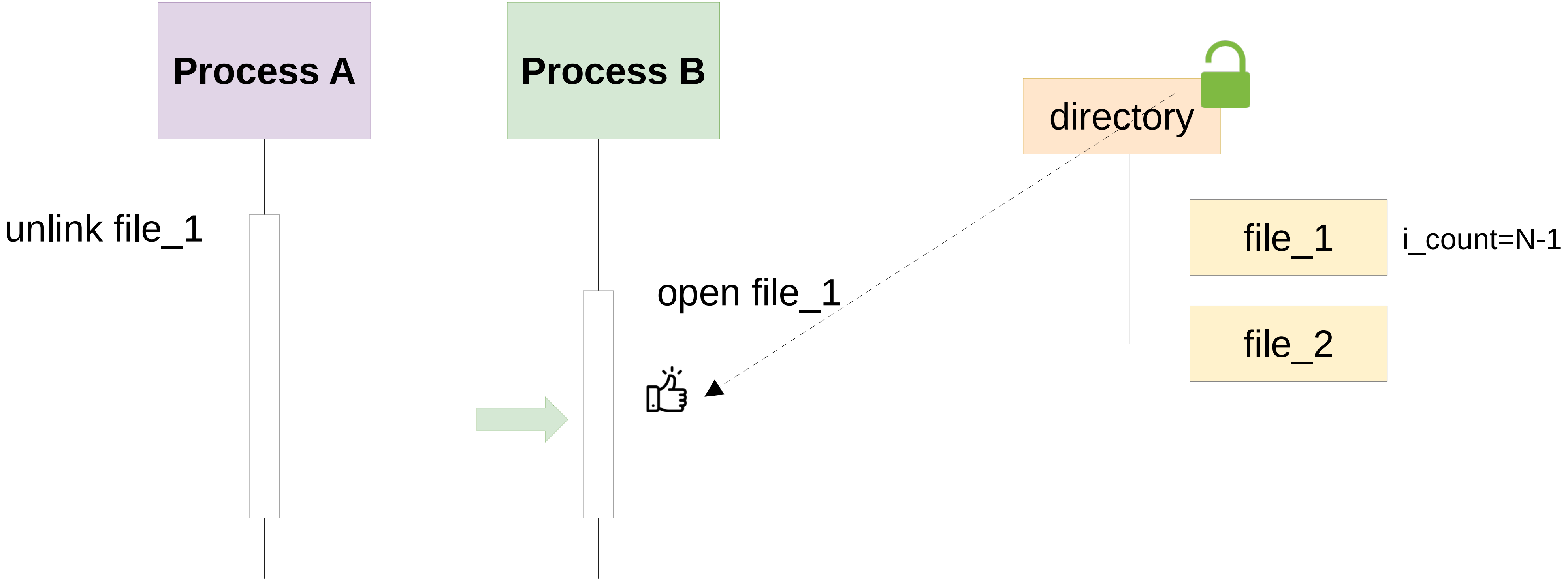
Exploitation - Example lock usage: removing a file

- Process A finishes the unlink



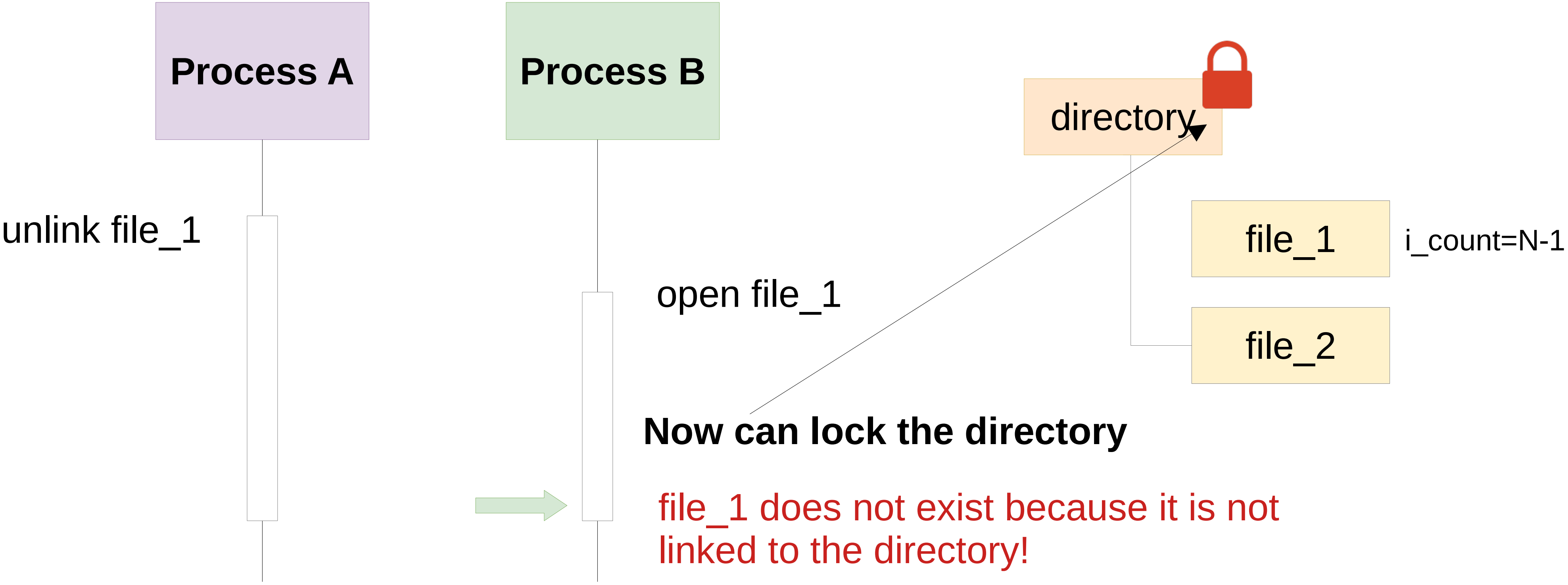
Exploitation - Example lock usage: removing a file

- **Process B is resumed**



Exploitation - Example lock usage: removing a file

- **Process B continues and returns an error**



Now can lock the directory

file_1 does not exist because it is not linked to the directory!

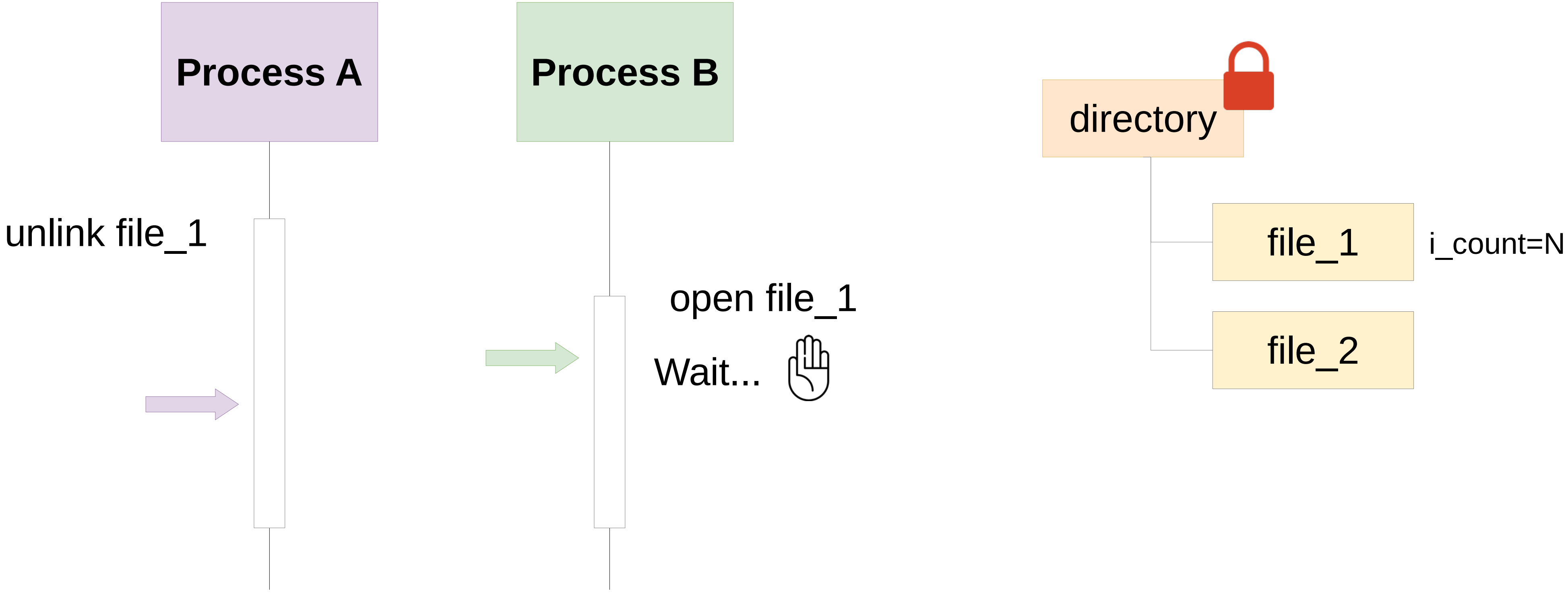
Exploitation - A reminder about the bug

- **If we perform an action which is not implemented (like *mkdir*) *shiftfs* will unlock the inode directory**
  - We can have several processes doing modifications in the same directory at the same time

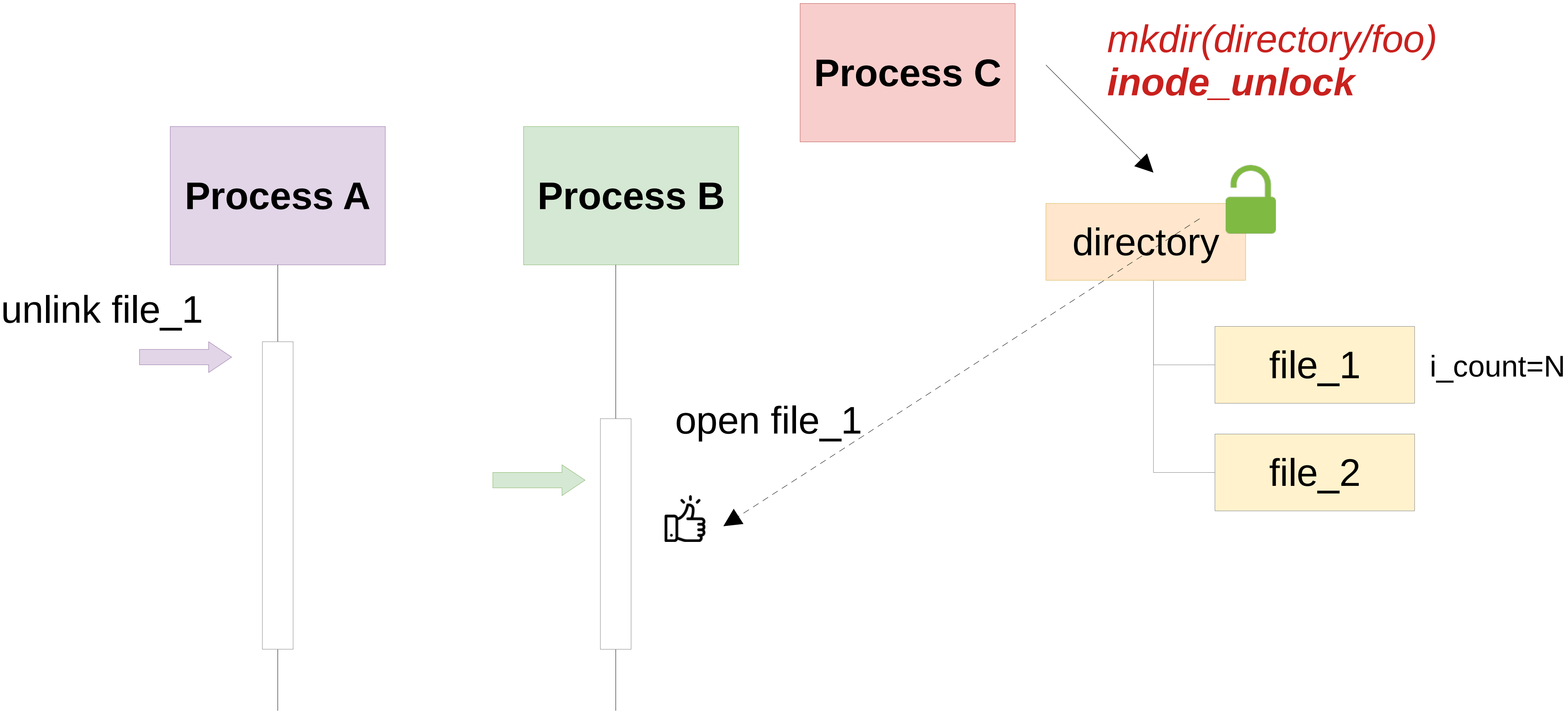
Exploitation - Example lock usage: removing a file

SYNACKTIV

- **Remember when Process B was waiting for the lock...**



Exploitation - Example lock usage: removing a file



Process B starts doing things whereas Process A did not finished

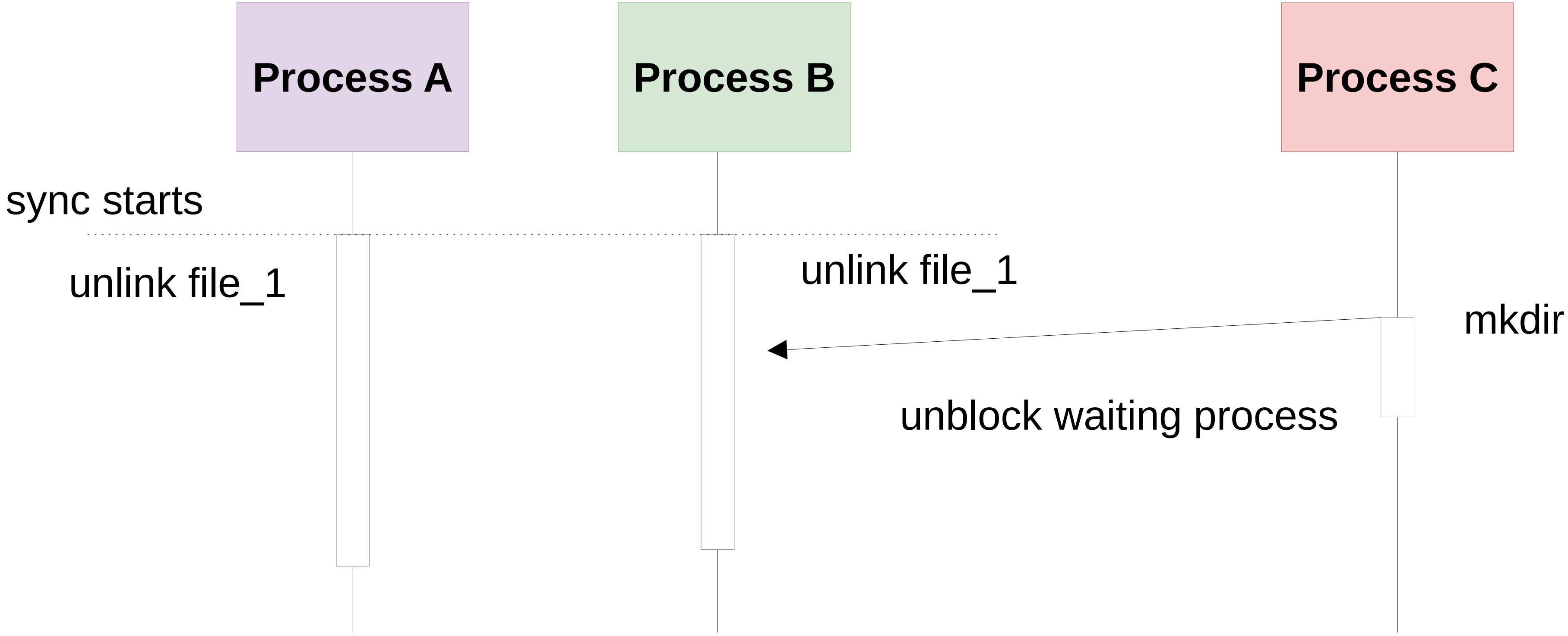
Exploitation - Taking advantage of the bug

- **During an unlink, the *i_count* value is decremented**
  - The reference due to the link with the directory inode is removed

    → During 2 simultaneous unlinks the *i_count* could be decremented twice
- **We can reach zero while the system is still using the inode**
  - The inode will be freed and in an Use-After-Free state

Exploitation - How to get an UAF ?

- **Exploit idea**


Process A

Process B

Process C

sync starts

unlink file_1

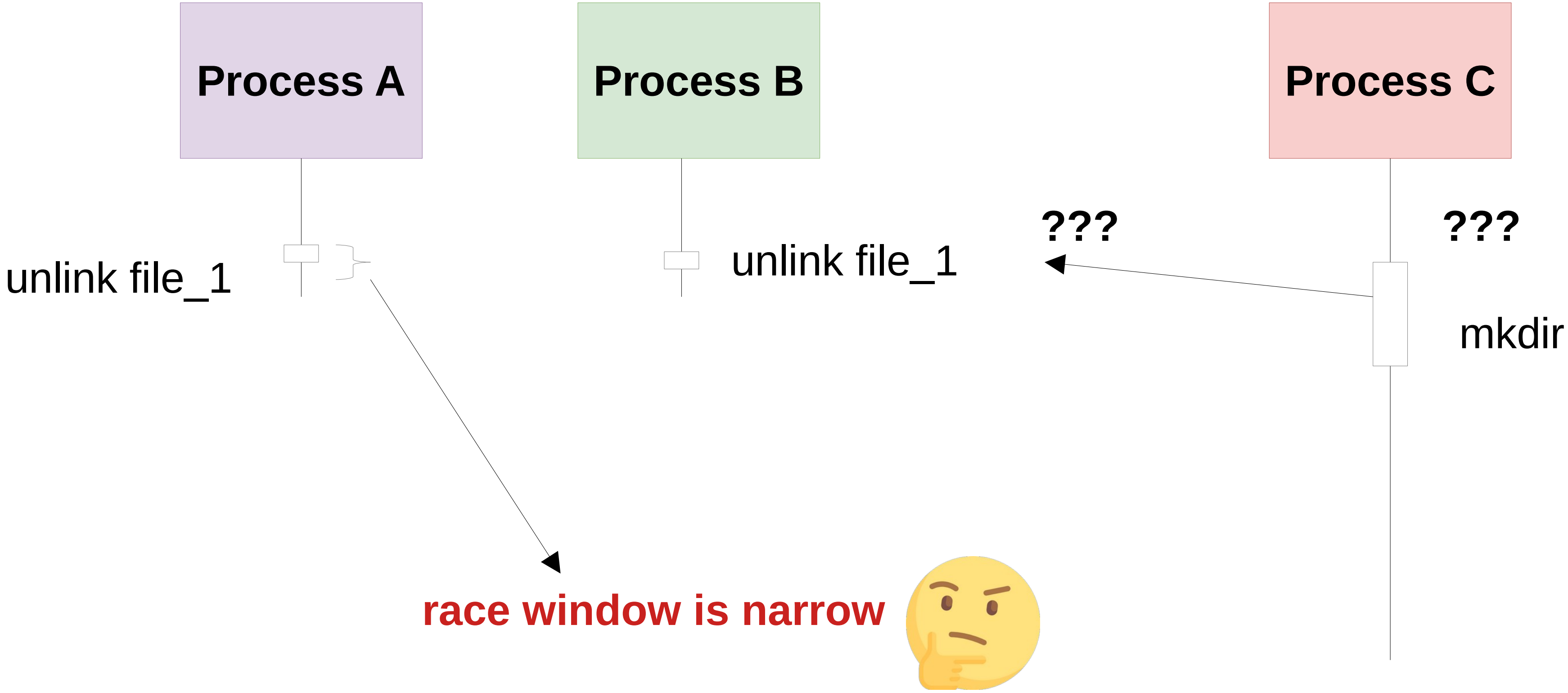
unlink file_1

mkdir

unblock waiting process

Exploitation - How to get an UAF ?

- **Reality...**



Exploitation - How to work on a race condition

SYNACKTIV

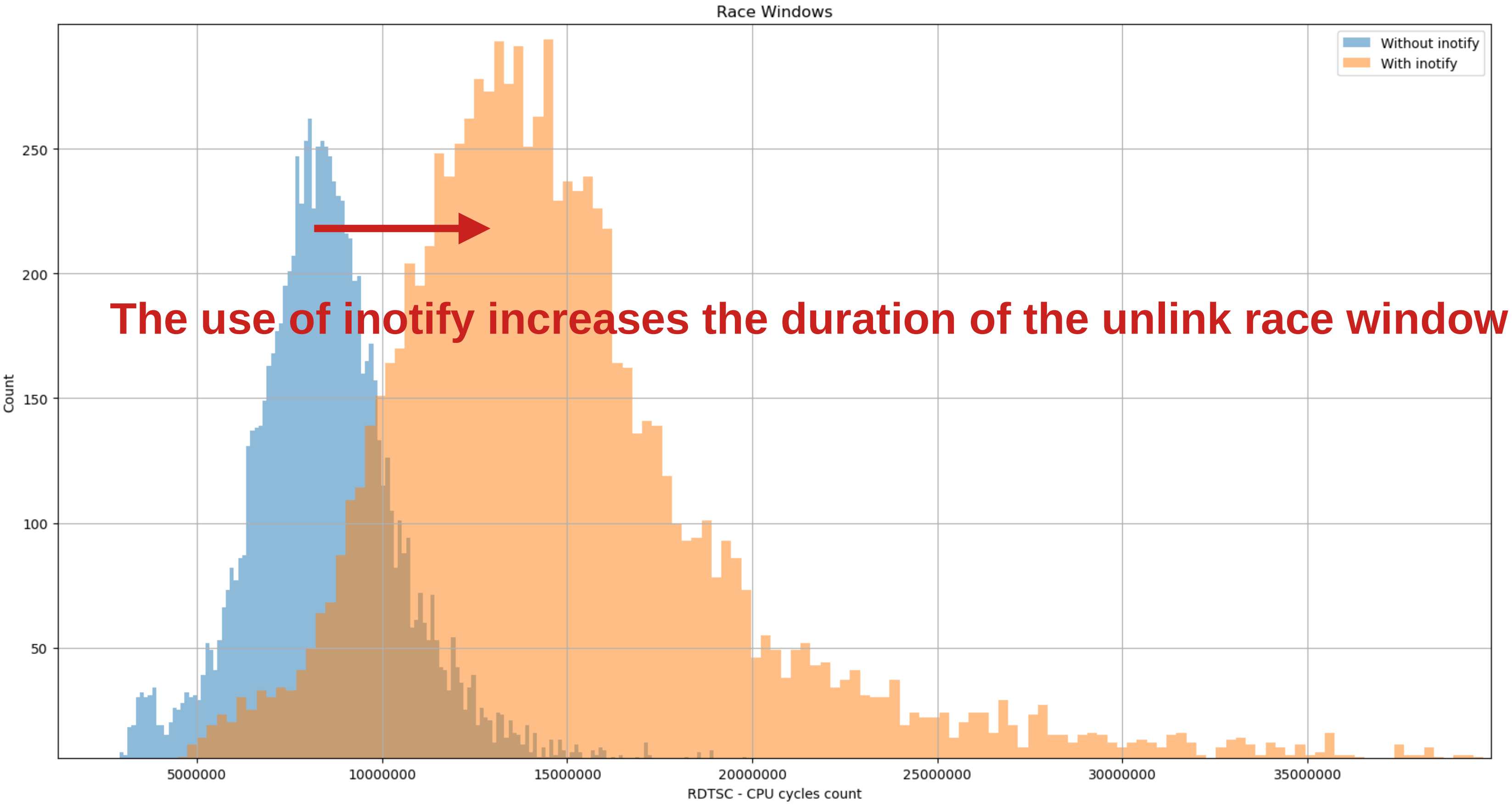
- **Work with a minimal setup**
    - Minimal but representative kernel in QEMU (same kernel configuration)
    - Be able to build and to test quickly
- **Some tips used**
    - Start to add a comfortable sleep to increase the race window.
        - The longer it takes, the easier it is to win the race!
    - Measure the timing to test your ideas using *rdtsc()*
    - Assign a process to a specific CPU and set its task priority.
- ***Kernel scheduler exploitation tricks and technical***
    - *Racing against the clock by Jann Horn (Google Project Zero)*
    - *ExpRace Academic Paper (Yoochan Lee, Changwoo Min, Byoungyoung Lee)*

Exploitation - Increasing the race window

- **By measuring the unlink race window, we observe that registering some inotify events increases the duration of the unlink operation!**
  - Without: mean 9258953 (on 10 000 tests)
  - With: mean 17359443 (+ ~90%)

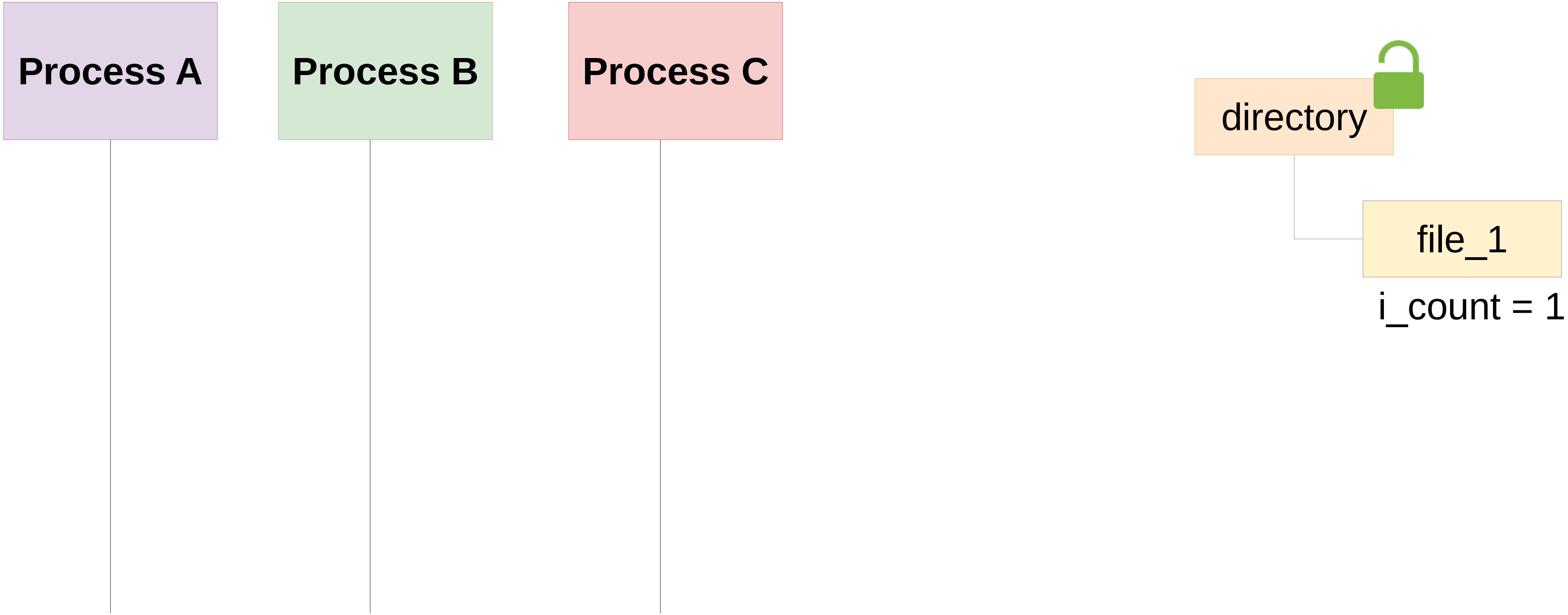
- **Prior to triggering the race, another process registers an inotify to receive notifications when a deletion occurs in the folder**
  - Success rate ~ 1/100 attempts (only takes few seconds)

Exploitation - Some Race window statistics

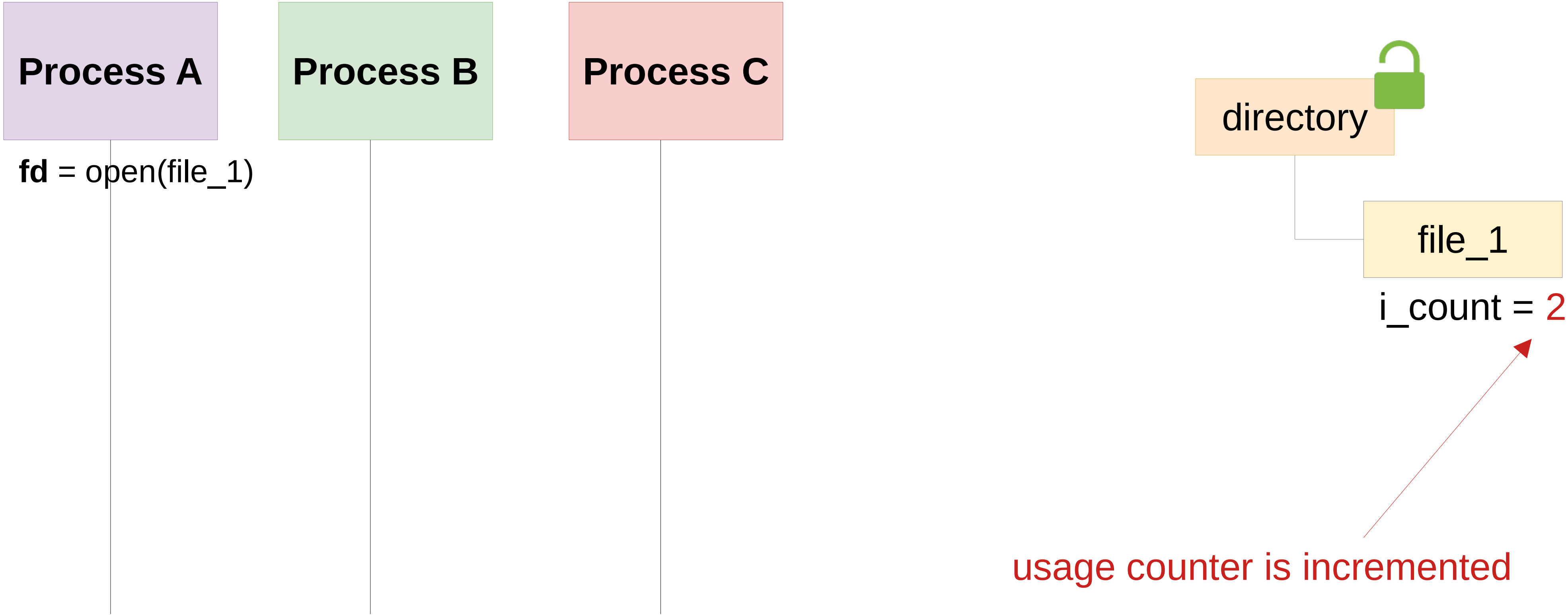


The use of inotify increases the duration of the unlink race window!

Exploitation - Winning the race



Exploitation - Winning the race


Process A

Process B

Process C

fd = open(file_1)

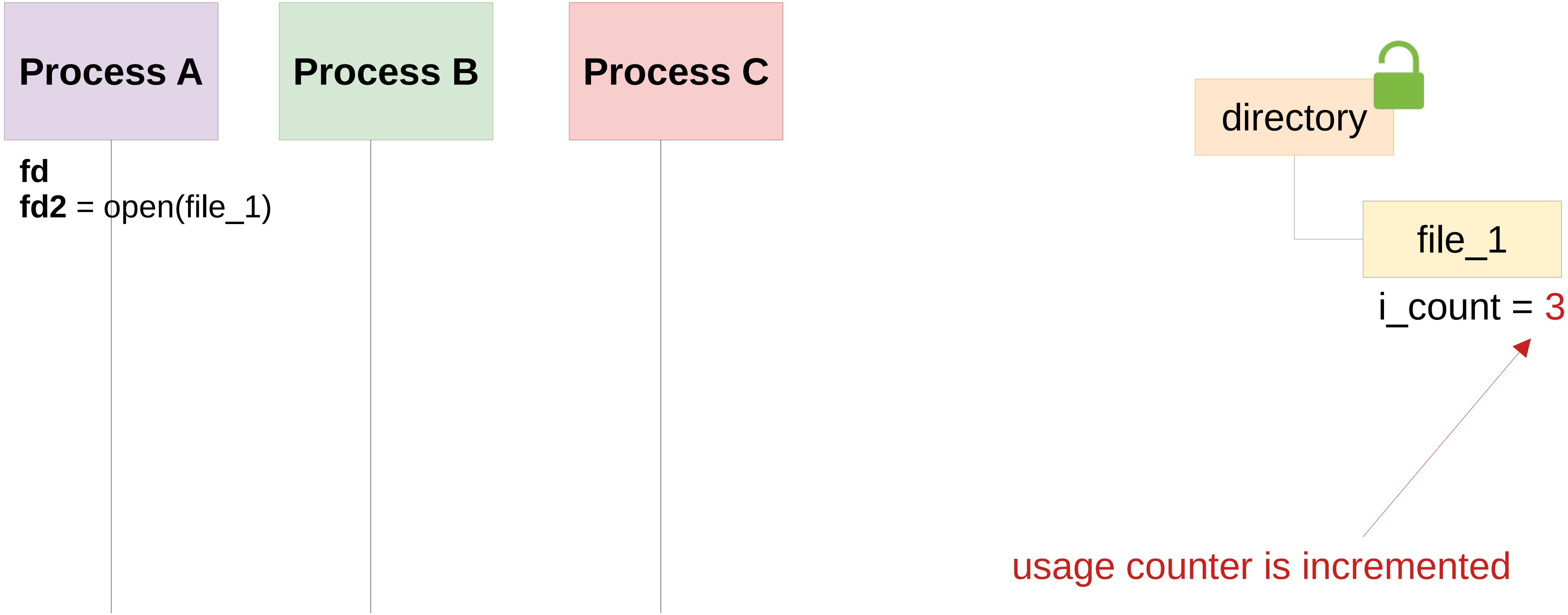
directory

file_1

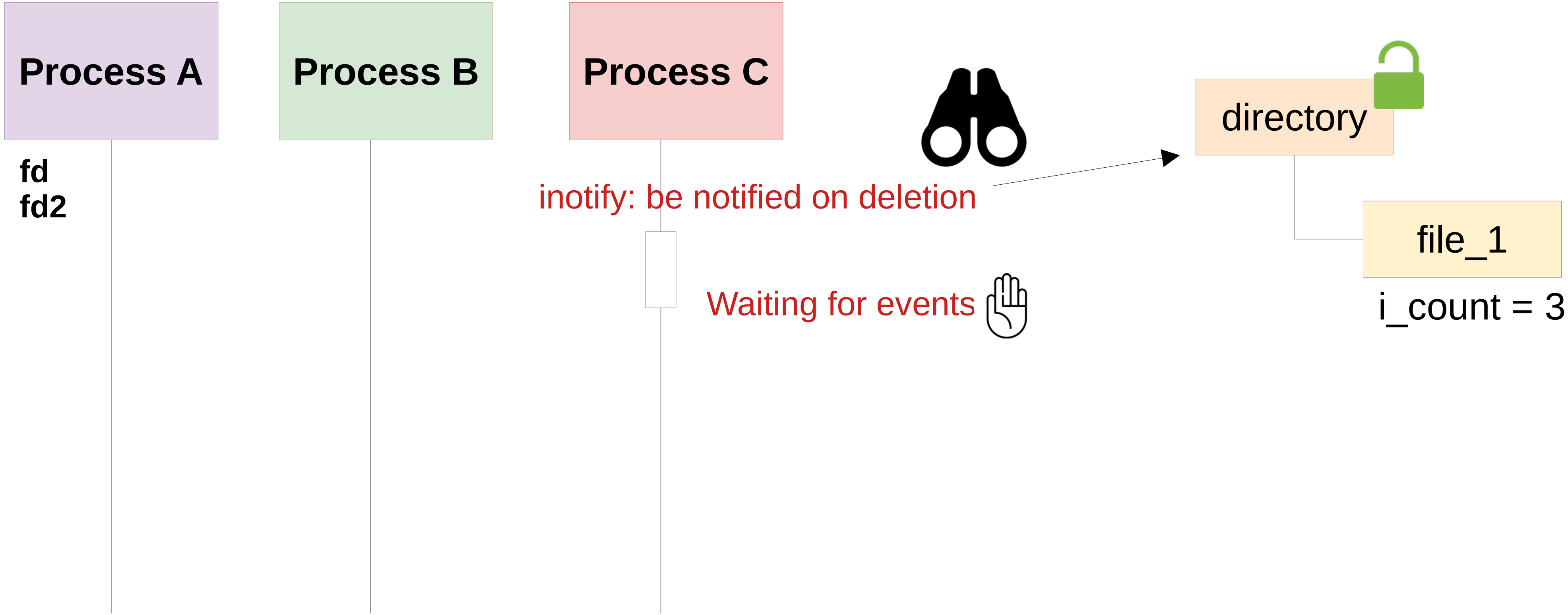
i_count = 2

usage counter is incremented

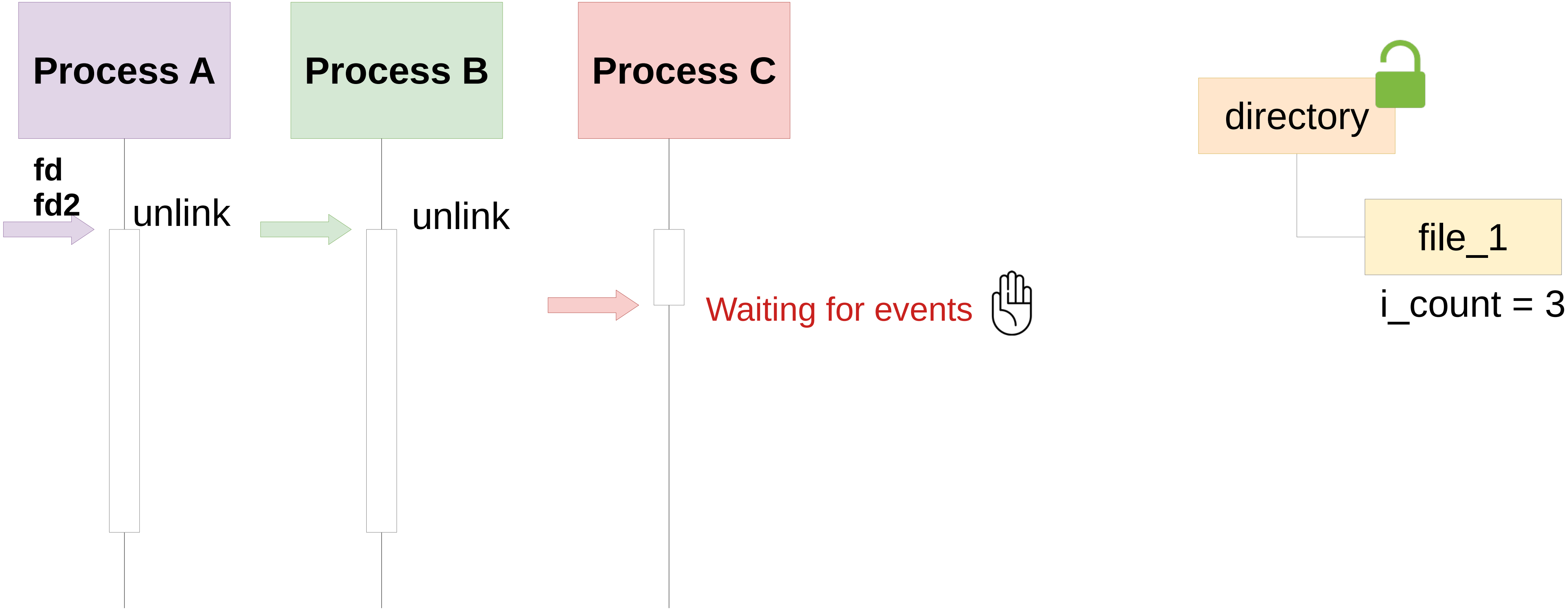
Exploitation - Winning the race



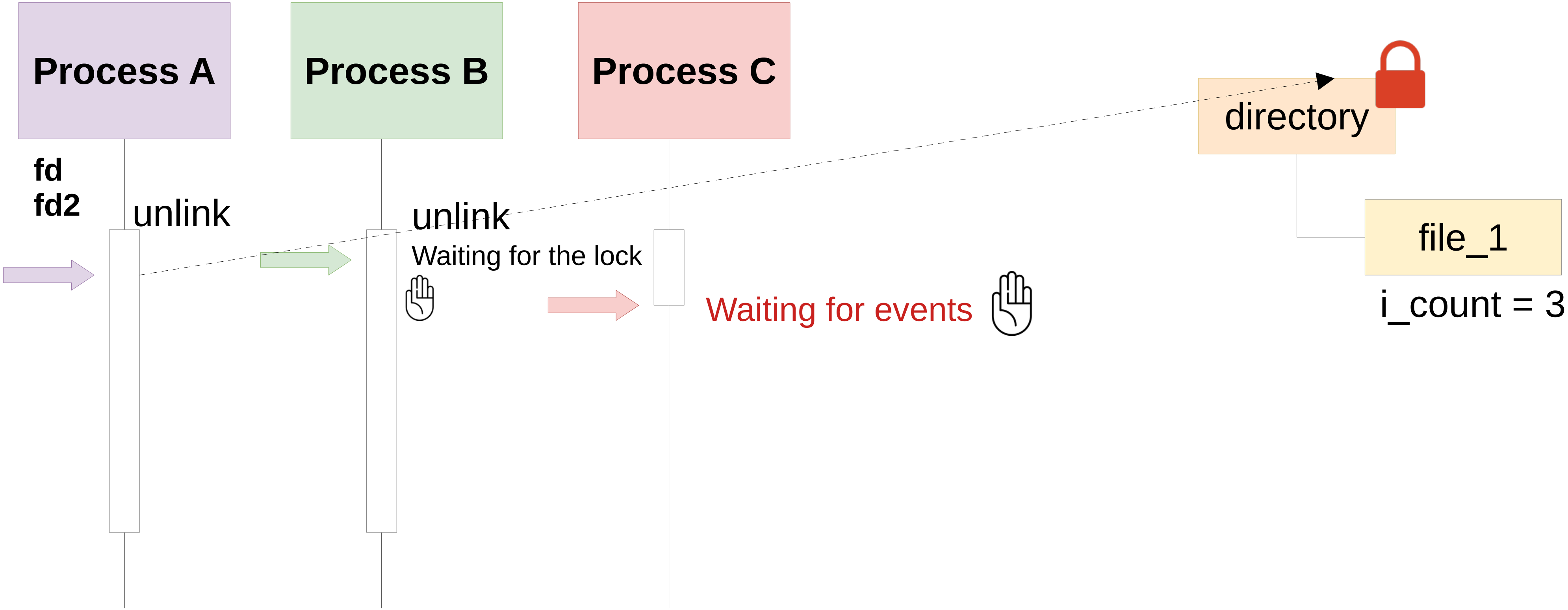
Exploitation - Winning the race



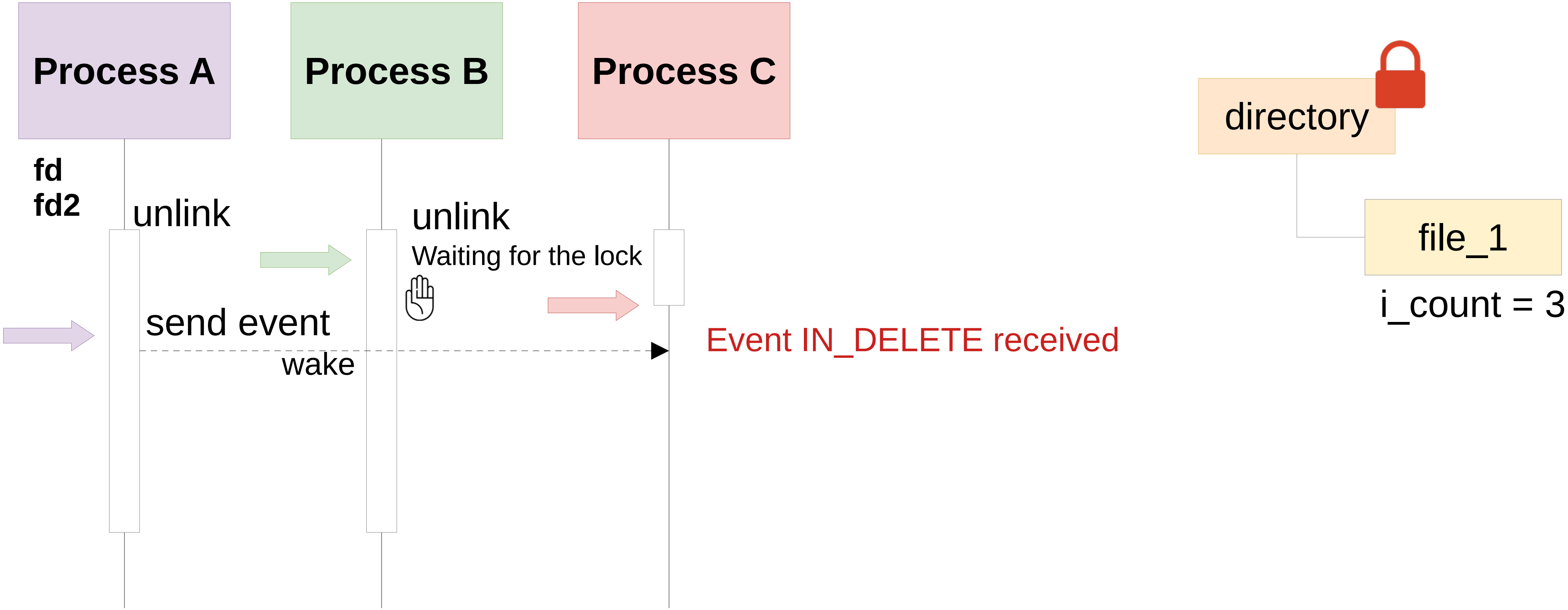
Exploitation - Winning the race



Exploitation - Winning the race



Exploitation - Winning the race



Process A

Process B

Process C

fd
fd2

unlink

unlink

Waiting for the lock

send event

wake

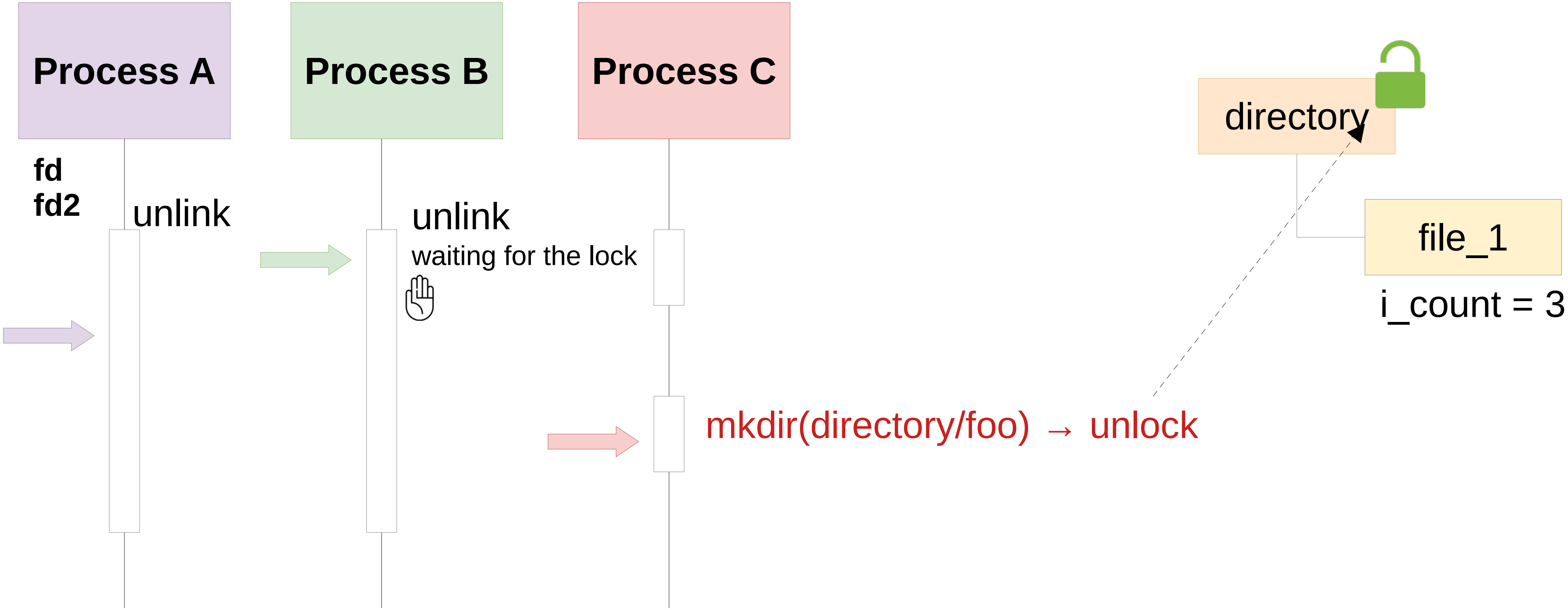
Event IN_DELETE received

directory

file_1

i_count = 3

Exploitation - Winning the race



Process A

Process B

Process C

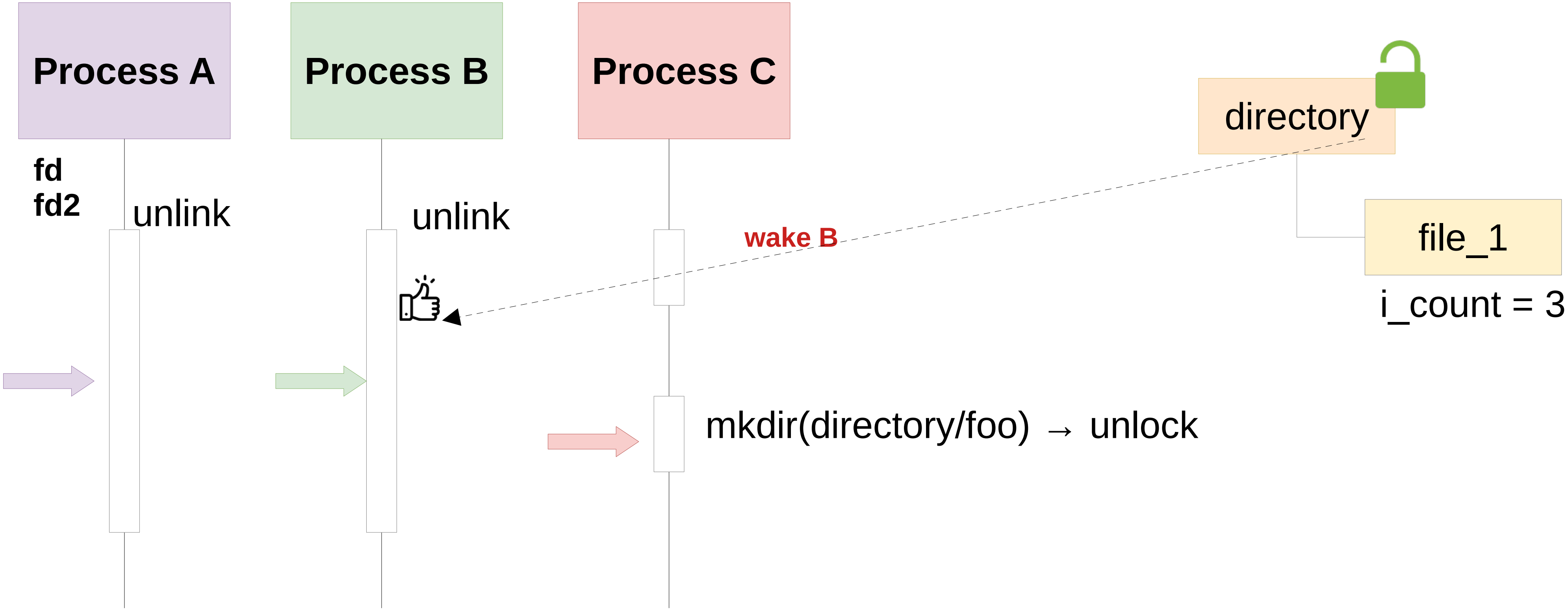
fd
fd2

unlink

unlink
waiting for the lock

mkdir(directory/foo) → unlock

Exploitation - Winning the race



Process A

Process B

Process C

directory

file_1

i_count = 3

fd
fd2

unlink

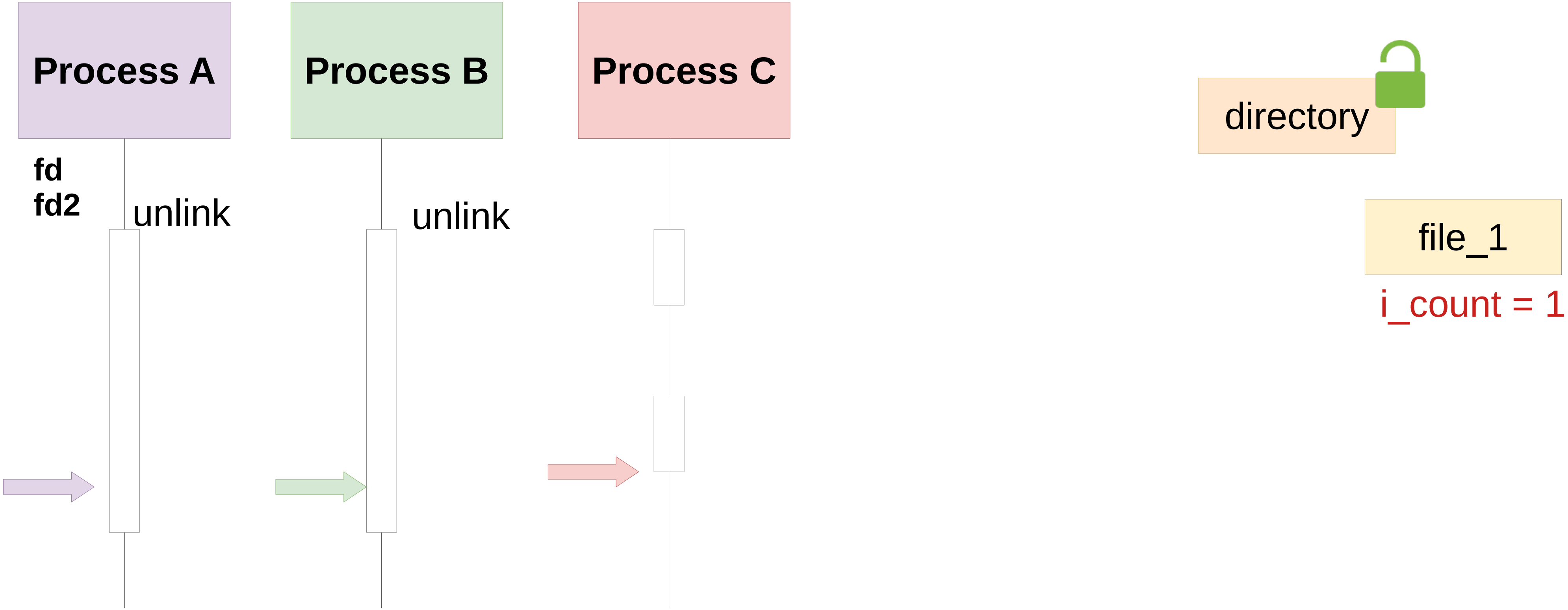
unlink

wake B

mkdir(directory/foo) → unlock

2 simultaneous unlinks → i_count will be decremented twice

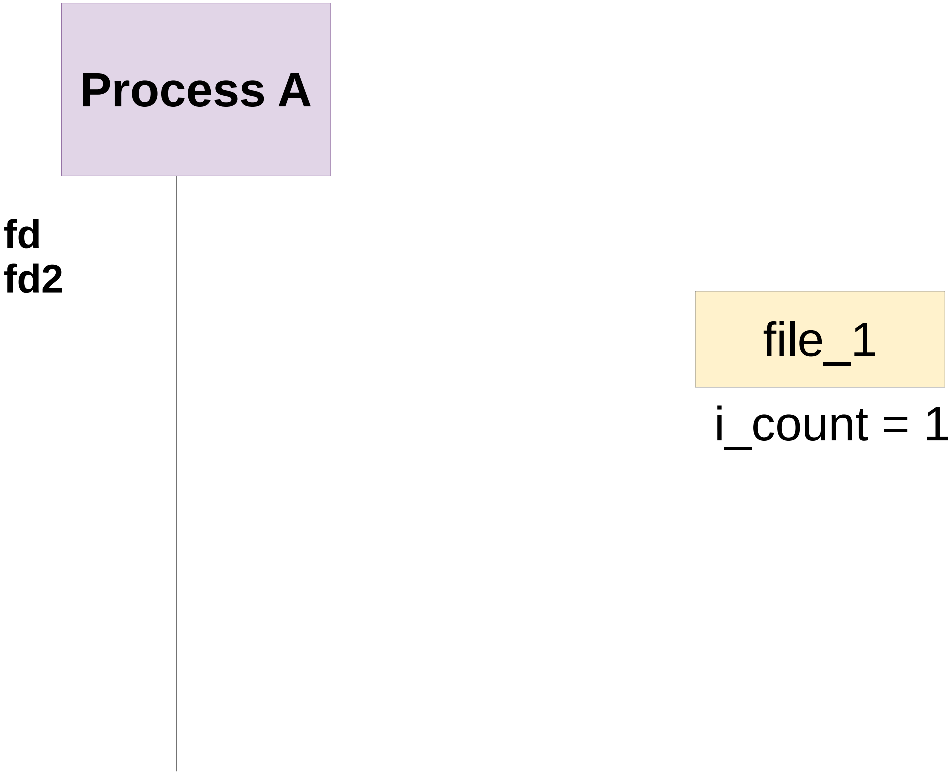
Exploitation - Winning the race



2 simultaneous unlinks → i_count will be decremented twice

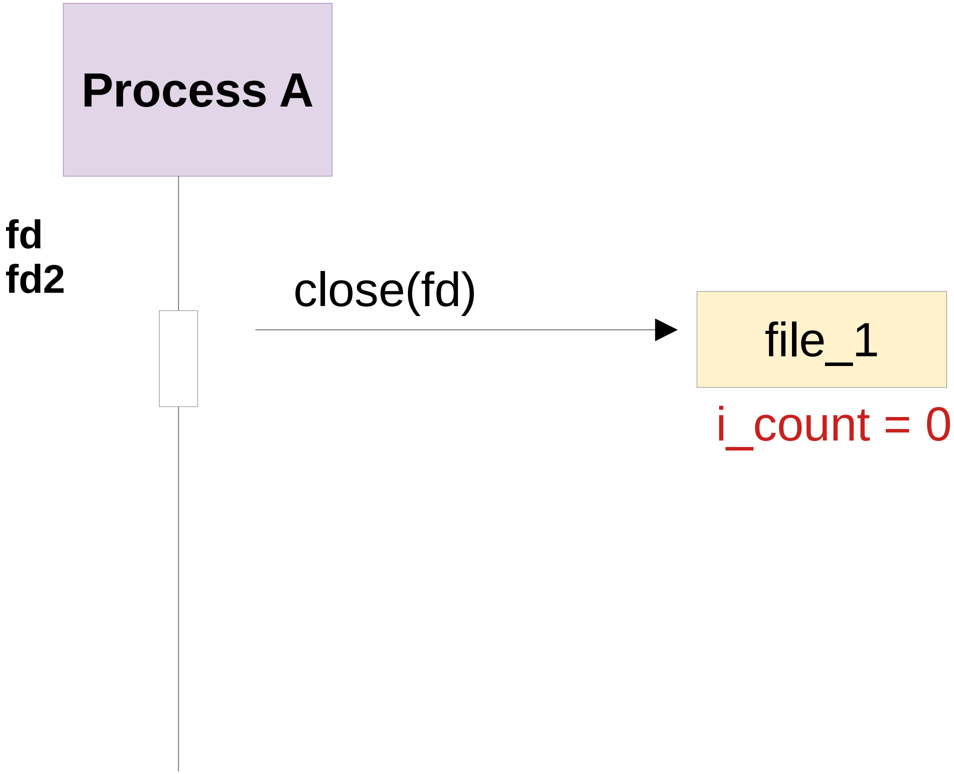
Exploitation - Winning the race

- **Process A still has file descriptors *fd* and *fd2* linked to file_1**
  - We can chose when *file_1* will be deleted by closing *fd*



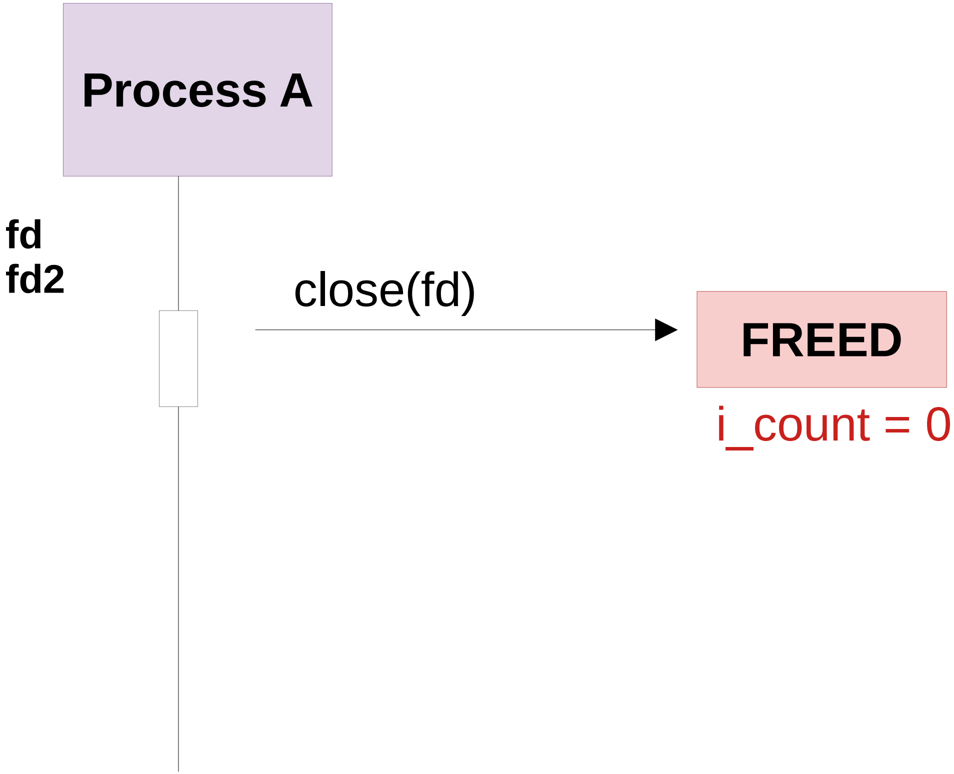
Exploitation - Winning the race

- **Process A still has file descriptors *fd* and *fd2* linked to file_1**
  - We can chose when *file_1* will be deleted by closing *fd*



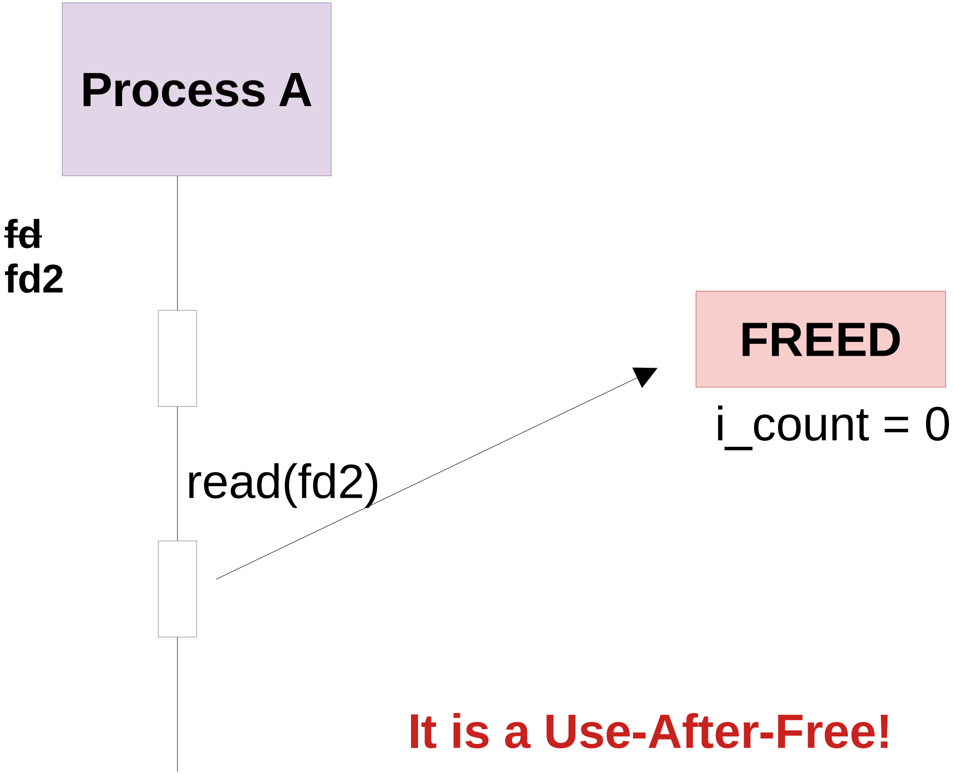
Exploitation - Winning the race

- **Process A still has file descriptors *fd* and *fd2* linked to file_1**
  - We can chose when *file_1* will be deleted by closing *fd*



Exploitation - Winning the race

- **Process A still has file descriptors *fd* and *fd2* linked to file_1**
  - We can chose when *file_1* will be deleted by closing *fd*
  - We can later access this file using the remaining FD

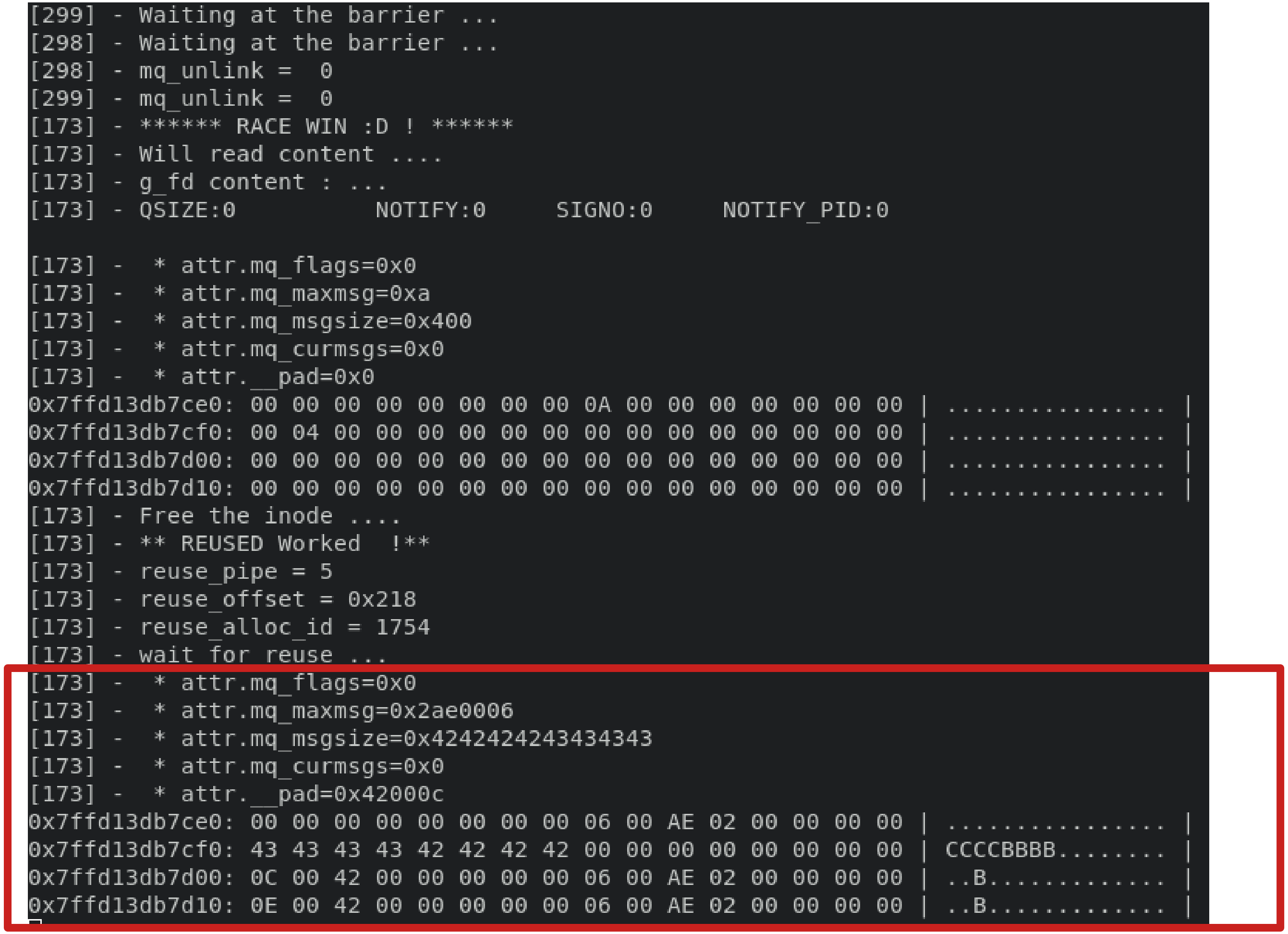


It is a Use-After-Free!

Exploitation - Exploiting the UAF

- **We are now in a classic Use-After-Free (UAF) situation**
- **There's no time to go into further details in this presentation :(**
- **All the exploitation steps are as follows**
  - Win the race to have a UAF ✓
  - Reuse the freed inode with the controlled data ✓
    - (not simple because the inode is in a dedicated slab ...)
  - Create kernel read/write primitives WORK IN PROGRESS
  - Leak a kernel pointer TODO
  - Patch the process credentials to elevate to root privileges TODO

Exploitation - Exploiting the UAF



```
[299] - Waiting at the barrier ...
[298] - Waiting at the barrier ...
[298] - mq_unlink =  0
[299] - mq_unlink =  0
[173] - ****** RACE WIN :D ! ******
[173] - Will read content ....
[173] - g_fd content : ...
[173] - QSIZE:0          NOTIFY:0     SIGNO:0     NOTIFY_PID:0

[173] -  * attr.mq_flags=0x0
[173] -  * attr.mq_maxmsg=0xa
[173] -  * attr.mq_msgsize=0x400
[173] -  * attr.mq_curmsgs=0x0
[173] -  * attr.__pad=0x0
0x7ffd13db7ce0: 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00 00 | ................ |
0x7ffd13db7cf0: 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................ |
0x7ffd13db7d00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................ |
0x7ffd13db7d10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................ |
[173] - Free the inode ....
[173] - ** REUSED Worked  !**
[173] - reuse_pipe = 5
[173] - reuse_offset = 0x218
[173] - reuse_alloc_id = 1754
[173] - wait for reuse ...
[173] -  * attr.mq_flags=0x0
[173] -  * attr.mq_maxmsg=0x2ae0006
[173] -  * attr.mq_msgsize=0x4242424243434343
[173] -  * attr.mq_curmsgs=0x0
[173] -  * attr.__pad=0x42000c
0x7ffd13db7ce0: 00 00 00 00 00 00 00 00 06 00 AE 02 00 00 00 00 | ................ |
0x7ffd13db7cf0: 43 43 43 43 42 42 42 42 00 00 00 00 00 00 00 00 | CCCCBBBB........ |
0x7ffd13db7d00: 0C 00 42 00 00 00 00 00 06 00 AE 02 00 00 00 00 | ..B............. |
0x7ffd13db7d10: 0E 00 42 00 00 00 00 00 06 00 AE 02 00 00 00 00 | ..B............. |
```

Reuse with arbitrary data !

Exploitation - Testing on the up to date Ubuntu

- **Trying the race on the up to date Ubuntu VM ...**
- **It did not work as expected**
  - If the exploit loses the race, the CPU is stuck!
  - Have only 1 try by CPU...

- **Why this behavior? The shiftfs code did not change!**

→ A patch in the kernel lock (commit d257cc8c)

Exploitation - I Gave up

- Winning a race with just one attempt by the CPU seems impossible... I gave up

Bug reported to Ubuntu

- Got the CVE-2023-2612 with the following patch

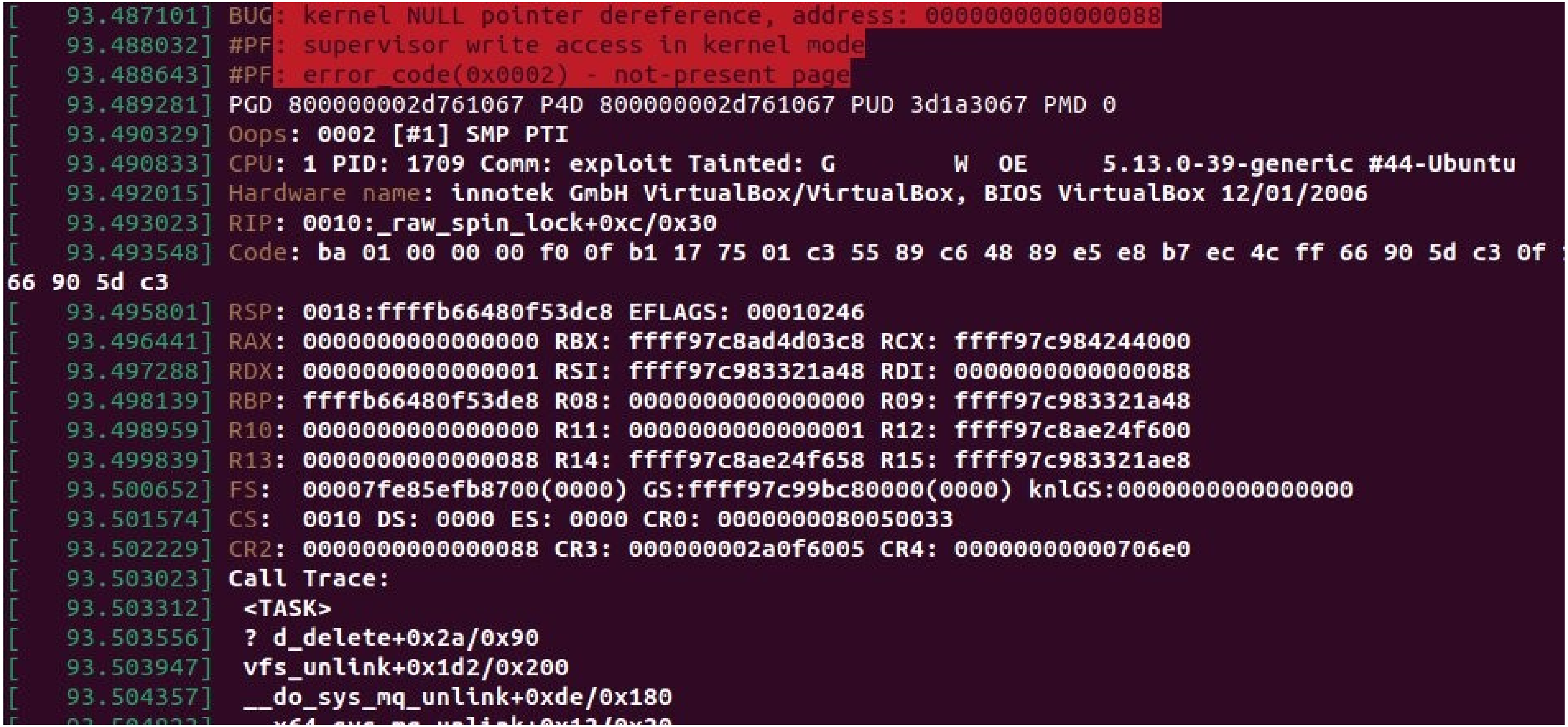
```
diff --git a/fs/shiftfs.c b/fs/shiftfs.c
index a76391c2246a..dab08fdd6638 100644
--- a/fs/shiftfs.c
+++ b/fs/shiftfs.c
@@ -409,6 +409,8 @@ static int shiftfs_create_object(struct inode *diri, struct dentry *dentry,
      const struct inode_operations *loweri_dir_iop = loweri_dir->i_op;
      struct dentry *lowerd_link = NULL;

+     inode_lock_nested(loweri_dir, I_MUTEX_PARENT);
+
      if (hardlink) {
            loweri_iop_ptr = loweri_dir_iop->link;
      } else {
@@ -434,8 +436,6 @@ static int shiftfs_create_object(struct inode *diri, struct dentry *dentry,
            goto out_iput;
      }

-     inode_lock_nested(loweri_dir, I_MUTEX_PARENT);
-
      if (!hardlink) {
            inode = new_inode(dir_sb);
            if (!inode) {
```

While preparing Grehack slides...

IDEA - Improve the race success

- **Using a process that uses inotify on the directory increases the race window**
- **What if several processes do the same?**
  - We can register up to 128 processes to monitor deletions in the directory
    - limited by */proc/sys/fs/inotify/max_user_instances*
  - This strategy significantly increases the success rate (by more than 50%)
- **The race condition can be won even with the kernel locking patch**

IDEA - Improve the race success



```
[   93.487101] BUG: kernel NULL pointer dereference, address: 0000000000000088
[   93.488032] #PF: supervisor write access in kernel mode
[   93.488643] #PF: error_code(0x0002) - not-present page
[   93.489281] PGD 800000002d761067 P4D 800000002d761067 PUD 3d1a3067 PMD 0
[   93.490329] Oops: 0002 [#1] SMP PTI
[   93.490833] CPU: 1 PID: 1709 Comm: exploit Tainted: G        W  OE     5.13.0-39-generic #44-Ubuntu
[   93.492015] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[   93.493023] RIP: 0010:_raw_spin_lock+0xc/0x30
[   93.493548] Code: ba 01 00 00 00 f0 0f b1 17 75 01 c3 55 89 c6 48 89 e5 e8 b7 ec 4c ff 66 90 5d c3 0f :
66 90 5d c3
[   93.495801] RSP: 0018:ffffb66480f53dc8 EFLAGS: 00010246
[   93.496441] RAX: 0000000000000000 RBX: ffff97c8ad4d03c8 RCX: ffff97c984244000
[   93.497288] RDX: 0000000000000001 RSI: ffff97c983321a48 RDI: 0000000000000088
[   93.498139] RBP: ffffb66480f53de8 R08: 0000000000000000 R09: ffff97c983321a48
[   93.498959] R10: 0000000000000000 R11: 0000000000000001 R12: ffff97c8ae24f600
[   93.499839] R13: 0000000000000088 R14: ffff97c8ae24f658 R15: ffff97c983321ae8
[   93.500652] FS:  00007fe85efb8700(0000) GS:ffff97c99bc80000(0000) knlGS:0000000000000000
[   93.501574] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   93.502229] CR2: 0000000000000088 CR3: 000000002a0f6005 CR4: 00000000000706e0
[   93.503023] Call Trace:
[   93.503312]  <TASK>
[   93.503556]  ? d_delete+0x2a/0x90
[   93.503947]  vfs_unlink+0x1d2/0x200
[   93.504357]  __do_sys_mq_unlink+0xde/0x180
[   93.504833]  __x64_sys_mq_unlink+0x12/0x20
```

Conclusion

- **A very interesting study, learned a lot about Linux VFS internals**
- **Namespaces are a very interesting attack surface**
  - Ubuntu plans to restrict them for Ubuntu 23.10
- **Do not give up too fast!**
  - Take a step back
  - There is perhaps a solution

References

SYNACKTIV

- **Vincent Dehors shiftfs exploitation (CVE-2021-3492)**
- **VFS documentation**
  - Linux VFS documentation
- **Exploit a race**
  - Racing against the clock by Jann Horn (Google Project Zero)
  - ExpRace Academic Paper (Yoochan Lee, Changwoo Min, Byoungyoung Lee)
- **Slab exploitation (page-level heap fengshui)**
  - https://etenal.me/archives/1825 (Xiaochen Zou and Zhiyun Qian)

SYNACKTIV

in https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com