Systematic Testing of Fault Handling Code in Linux Kernel

Alexey Khoroshilov
Andrey Tsyvarev

Institute for System Programming of the Russian Academy of Sciences
Fault Handling Code

821    error = filemap_write_and_wait_range(VFS_I(ip)->i_mapping,
822                        ip->i_d.di_size, newsize);
823    if (error)
824        return error;
...
852    tp = xfs_trans_alloc(mp, XFS_TRANS_SETATTR_SIZE);
853    error = xfs_trans_reserve(tp, &M_RES(mp)->tr_itruncate, 0, 0);
854    if (error)
855        goto out_trans_cancel;
...
925 out_unlock:
926    if (lock_flags)
927        xfs_iunlock(ip, lock_flags);
928    return error;
929
930 out_trans_abort:
931    commit_flags |= XFS_TRANS_ABORT;
932 out_trans_cancel:
933    xfs_trans_cancel(tp, commit_flags);
934    goto out_unlock;
Fault Handling Code

DOING WHAT YOU LIKE IS FREEDOM
Fault Handling Code

DOING
WHAT YOU LIKE
IS FREEDOM

LIKING
WHAT YOU DO
IS HAPPINESS
Fault Handling Code

```c
821     error = filemap_write_and_wait_range(VFS_I(ip)->i_mapping,
822                                        ip->i_d.di_size, newsize);
823     if (error)
824         return error;
...
852     tp = xfs_trans_alloc(mp, XFS_TRANS_SETATTR_SIZE);
853     error = xfs_trans_reserve(tp, &M_RES(mp)->tr_itruncate, 0, 0);
854     if (error)
855         goto out_trans_cancel;
...
925 out_unlock:
926     if (lock_flags)
927         xfs_iunlock(ip, lock_flags);
928     return error;
929
930 out_trans_abort:
931     commit_flags |= XFS_TRANS_ABORT;
932 out_trans_cancel:
933     xfs_trans_cancel(tp, commit_flags);
934     goto out_unlock;
```
Fault Handling Code

- Is not so fun
- Is really hard to keep all details in mind
Fault Handling Code

- Is not so fun
- Is really hard to keep all details in mind
- Practically is not tested
- Is hard to test even if you want to
Fault Handling Code

- Is not so fun
- Is really hard to keep all details in mind
- Practically is not tested
- Is hard to test even if you want to
- Bugs seldom(never) occurs

=> low pressure to care
Why do we care?

- It beats someone time to time
- Safety critical systems
- Certification authorities
How to improve situation?

• Managed resources
  + No code, no problems
    – Limited scope

• Static analysis
  + Analyzes all paths at once
    – Detects prescribed set of consequences (mostly local)
    – False alarms

• Run-time testing
  + Detects even hidden consequences
  + Almost no false alarms
    – Tests are needed
    – Specific hardware may be needed (for drivers testing)
Run-Time Testing of Fault Handling

- Manually targeted test cases
  - The highest quality
  - Expensive to develop and to maintain
  - Not scalable

- Random fault injection on top of existing tests
  - Cheap
  - Oracle problem
  - No any guarantee
  - When to finish?
Systematic Approach

- **Hypothesis:**
  - Existing test lead to deterministic control flow in kernel code

- **Idea:**
  - Execute existing tests and collect all potential fault points in kernel code
  - Systematically enumerate the points and inject faults there
Experiments - Outline

- Target code
- Fault injection implementation
- Methodology
- Results
Experiments - Target

- Target code: file system drivers
- Reasons:
  - Failure handling is more important than in average
    - Potential data loss, etc.
  - Same tests for many drivers
  - It does not require specific hardware
  - Complex enough
Linux File System Layers

User Space Application

sys_mount, sys_open, sys_read, ...

Block Based FS:
- ext4, xfs, btrfs, jfs, ...

Network FS:
- nfs, coda, gfs, ocfs, ...

Pseudo FS:
- proc, sysfs, ...

Special Purpose:
- tmpfs, ramfs, ...

Block I/O layer
- Optional stackable devices (md, dm, ...)
- I/O schedulers

Direct I/O

Buffer cache / Page cache

ioctl, sysfs

Disk

Block Driver

Block Driver

CD
# File System Drivers - Size

<table>
<thead>
<tr>
<th>File System Driver</th>
<th>Size, LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFS</td>
<td>18 KLOC</td>
</tr>
<tr>
<td>Ext4</td>
<td>37 KLoC with jbd2</td>
</tr>
<tr>
<td>XFS</td>
<td>69 KLoC</td>
</tr>
<tr>
<td>BTRFS</td>
<td>82 KLoC</td>
</tr>
<tr>
<td>F2FS</td>
<td>12 KLoC</td>
</tr>
</tbody>
</table>
File System Driver - VFS Interface

- file_system_type
- super_operations
- export_operations
- inode_operations
- file_operations
- vm_operations
- address_space_operations
- dquot_operations
- quotactl_ops
- dentry_operations

~100 interfaces in total
## FS Driver - Userspace Interface

<table>
<thead>
<tr>
<th>File System Driver</th>
<th>ioctl</th>
<th>sysfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFS</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Ext4</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>XFS</td>
<td>48</td>
<td>-</td>
</tr>
<tr>
<td>BTRFS</td>
<td>57</td>
<td>-</td>
</tr>
</tbody>
</table>
## FS Driver - Partition Options

<table>
<thead>
<tr>
<th>File System Driver</th>
<th>mount options</th>
<th>mkfs options</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFS</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>Ext4</td>
<td>50</td>
<td>~30</td>
</tr>
<tr>
<td>XFS</td>
<td>37</td>
<td>~30</td>
</tr>
<tr>
<td>BTRFS</td>
<td>36</td>
<td>8</td>
</tr>
</tbody>
</table>
FS Driver - On-Disk State

File System Hierarchy
* File Size
* File Attributes
* File Fragmentation
* File Content (holes,...)
FS Driver - In-Memory State

- Page Cache State
- Buffers State
- Delayed Allocation
- ...

...
Linux File System Layers

- User Space Application
- VFS
  - Block Based FS: ext4, xfs, btrfs, jfs, ...
  - Network FS: nfs, coda, gfs, ocfs, ...
  - Pseudo FS: proc, sysfs, ...
  - Special Purpose: tmpfs, ramfs, ...

- Direct I/O
- Block I/O layer
  - Optional stackable devices (md, dm, ...)
  - I/O schedulers

- FS Driver State
- VFS State*
- Buffer cache / Page cache

- File System State
- 100 interfaces
- 30-50 interfaces
- ioctl, sysfs

- 30 mount opts
- 30 mkfs opts

- Block Driver
- Disk

- CD
FS Driver - Fault Handling

- Memory Allocation Failures
- Disk Space Allocation Failures
- Read/Write Operation Failures
Fault Injection - Implementation

• Based on KEDR framework*
  • intercept requests for memory allocation/bio requests
    • to collect information about potential fault points
    • to inject faults
  • also used to detect memory/resources leaks

(*) http://linuxtesting.org/project/kedr
KEDR Workflow

http://linuxtesting.org/project/kedr
Experiments - Tests

- 10 deterministic tests from xfstests*
  - generic/
    - 001-003, 015, 018, 020, 053
  - ext4/
    - 002, 271, 306
- Linux File System Verification** tests
  - 180 unit tests for FS-related syscalls / ioctls
  - mount options iteration

(*) git://oss.sgi.com/xfs/cmds/xfstests
(**) http://linuxtesting.org/spruce
Experiments - Oracle Problem

• Assertions in tests are disabled
• Kernel oops/bugs detection
• Kernel assertions, lockdep, memcheck, etc.
• KEDR Leak Checker
Experiments - Methodology

• Collect source code coverage of FS driver on existing tests
• Collect source code coverage of FS driver on existing tests with fault simulation
• Measure an increment
Methodology - The Problem

- If kernel crashes code, coverage results are unreliable
Methodology - The Problem

- If kernel crashes code, coverage results are unreliable
- As a result
  - Ext4 was analyzed only
  - XFS, BTRF, JFS, F2FS, UbiFS, JFFS2 crashes and it is too labor and time consuming to collect reliable data
Experiment Results
Systematic Approach

• Hypothesis:
  • Existing test lead to deterministic control flow in kernel code

• Idea:
  • Execute existing tests and collect all potential fault points in kernel code
  • Systematically enumerate the points and inject faults there
## Complete Enumeration

<table>
<thead>
<tr>
<th>Test Description</th>
<th>Fault points</th>
<th>Expected Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xfstests (10 system tests)</td>
<td>270 327</td>
<td>2.5 years</td>
</tr>
<tr>
<td>LFSV (180 unit tests*76 mount options)</td>
<td>488 791</td>
<td>7 months</td>
</tr>
</tbody>
</table>
Possible Idea

- Unit test structure
  - Preamble
  - Main actions
  - Checks
  - Postamble

- What if account fault points inside main actions?
## Complete Enumeration

<table>
<thead>
<tr>
<th></th>
<th>Fault points</th>
<th>Expected Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xfstests (10 system tests)</td>
<td>270 327</td>
<td>2,5 years</td>
</tr>
<tr>
<td>LFSV (180 unit tests)</td>
<td>488 791</td>
<td>7 months</td>
</tr>
<tr>
<td>LFSV (180 unit tests) – main part only</td>
<td>9 226</td>
<td>1,5 hours</td>
</tr>
</tbody>
</table>

- that gives 311 new lines of code covered
- i.e. 18 seconds per line
Another Idea

• Automatic filtering
  • e.g. by Stack Trace of fault point
## LFSV Tests

<table>
<thead>
<tr>
<th></th>
<th>Increment new lines</th>
<th>Time min</th>
<th>Cost seconds/line</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFSV without fault simulation</td>
<td>-</td>
<td>110</td>
<td>-</td>
</tr>
<tr>
<td>LFSV – main only – no filter</td>
<td>311</td>
<td>92</td>
<td>18</td>
</tr>
<tr>
<td>LFSV – main only – stack filter</td>
<td>266</td>
<td>2</td>
<td>0.45</td>
</tr>
<tr>
<td>LFSV – whole test – no filter</td>
<td>unfeasible</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFSV – whole test – stack filter</td>
<td>333</td>
<td>4</td>
<td>0.72</td>
</tr>
</tbody>
</table>
Main-only vs. Whole

+ 2-3 times more cost effective
  - Manual work =>
    • expensive
    • error-prone
    • unscalable

+ More scalable
+ Better coverage
## Unit tests vs. System tests

<table>
<thead>
<tr>
<th></th>
<th>Increment new lines</th>
<th>Time min</th>
<th>Cost second/line</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFSV – whole test – stack filter</td>
<td>333</td>
<td>4</td>
<td>0.72</td>
</tr>
<tr>
<td>LFSV – whole test – stackset filter</td>
<td>354</td>
<td>9</td>
<td>1.53</td>
</tr>
<tr>
<td>Xfstests – stack filter</td>
<td>423</td>
<td>90</td>
<td>13</td>
</tr>
<tr>
<td>Xfstests – stackset filter</td>
<td>451</td>
<td>237</td>
<td>31</td>
</tr>
</tbody>
</table>

+ 10-30 times more cost effective

+ Better coverage
## Systematic vs. Random

<table>
<thead>
<tr>
<th>Test Description</th>
<th>Increment new lines</th>
<th>Time min</th>
<th>Cost second/line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xfstests without fault simulation</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Xfstests+random((p=0.01,\text{repeat}=200))</td>
<td>380</td>
<td>152</td>
<td>24</td>
</tr>
<tr>
<td>Xfstests+random((p=0.02,\text{repeat}=200))</td>
<td>373</td>
<td>116</td>
<td>19</td>
</tr>
<tr>
<td>Xfstests+random((p=0.05,\text{repeat}=200))</td>
<td>312</td>
<td>82</td>
<td>16</td>
</tr>
<tr>
<td>Xfstests+random((p=0.01,\text{repeat}=400))</td>
<td>451</td>
<td>350</td>
<td>47</td>
</tr>
<tr>
<td>Xfstests+stack filter</td>
<td>423</td>
<td>90</td>
<td>13</td>
</tr>
<tr>
<td>Xfstests+stackset filter</td>
<td>451</td>
<td>237</td>
<td>31</td>
</tr>
</tbody>
</table>
Systematic vs. Random

+ 2 times more cost effective
+ Repeatable results
  - Requires more complex engine
+ Cover double faults
  - Unpredictable
  - Nondeterministic
<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0010</td>
<td>Crash</td>
<td>f2fs: Possible use-after-free when umount filesystem</td>
<td>2014-09-08</td>
<td><a href="https://lkml.org/lkml/2014/7/21/198">https://lkml.org/lkml/2014/7/21/198</a> commit</td>
<td>Fixed in kernel 3.17-rc1</td>
</tr>
<tr>
<td>F0009</td>
<td>Crash</td>
<td>ext4: Destruction of ext4_groupinfo_caches during one mount causes BUG_ON for other mounted ext4 filesystems</td>
<td>2014-05-12</td>
<td><a href="https://lkml.org/lkml/2014/5/12/147">https://lkml.org/lkml/2014/5/12/147</a> commit</td>
<td>Fixed in kernel 3.16-rc1</td>
</tr>
<tr>
<td>F0008</td>
<td>Crash</td>
<td>f2fs: BUG_ON() is triggered in recover_inode_page() when mount valid f2fs filesystem</td>
<td>2014-04-18</td>
<td><a href="https://lkml.org/lkml/2014/4/14/189">https://lkml.org/lkml/2014/4/14/189</a> commit</td>
<td>Fixed in kernel 3.17-rc1</td>
</tr>
<tr>
<td>F0007</td>
<td>Crash</td>
<td>f2fs: f2fs unmount hangs if f2fs_init_acl() fails during mkdir syscall</td>
<td>2014-02-17</td>
<td><a href="https://lkml.org/lkml/2014/2/6/18">https://lkml.org/lkml/2014/2/6/18</a> commit</td>
<td>Fixed in kernel 3.15-rc1</td>
</tr>
<tr>
<td>F0006</td>
<td>Deadlock</td>
<td>f2fs: a deadlock in mkdir if ACL is enabled</td>
<td>2013-10-28</td>
<td><a href="https://lkml.org/lkml/2013/10/26/163">https://lkml.org/lkml/2013/10/26/163</a> commit</td>
<td>Fixed in kernel 3.12-rc3</td>
</tr>
<tr>
<td>F0001</td>
<td>Crash</td>
<td>ext4: NULL pointer dereference in mount_fs() because of ext4_fill_super() wrongly reports success</td>
<td>2012-11-08</td>
<td><a href="https://bugzilla.kernel.org/show_bug.cgi?id=48431">https://bugzilla.kernel.org/show_bug.cgi?id=48431</a> commit</td>
<td>Fixed in kernel 3.8-rc1</td>
</tr>
</tbody>
</table>
XFS bug

[ 3143.894108] Start: fsim.spruce.common.-L_SPRUCE_XFS:.utime.UtimeNormalNotNull.kmalloc.2
[ 3143.894110] KEDR FAULT SIMULATION: forcing a failure
[ 3143.894116] CPU: 0 PID: 7127 Comm: fs-driver-tests Tainted: G        W  O   3.17-generic #1
[ 3143.894118] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[ 3143.894119] ffff8800701e7be0 ffff8800701e7b98 ffffffff8169b4a1 0000000000000001
[ 3143.894185] ffff8800701e7bc8 ffffffffa022fbcf ffffffffa022fb25 ffff8800701e7fd8
[ 3143.894188] ffff880082421a80 00000000000008250 ffff8800701e7c18 ffffffffffa023f610
[ 3143.894190] Call Trace:
[ 3143.894197] [<ffffffff8169b4a1>] dump_stack+0x4d/0x66
[ 3143.894202] [<ffffffffa022fbcf>] kedr_fsim_point_simulate+0xaf/0xc0 [kedr_fault_simulation]
[ 3143.894208] [<ffffffffa023f610>] kedr_repl_kmem_cache_alloc+0x40/0x90 [kedr_fsm_cm]
[ 3143.894230] [<ffffffffa02401d3>] kedr_intermediate_func_kmem_cache_alloc+0x73/0xd0 [kedr_fsm_cm]
[ 3143.894264] [<ffffffffa038b497>] kmem_zone_alloc+0x77/0x100 [xfs]
[ 3143.894277] [<ffffffffa038fb7d>] xlog_ticket_alloc+0x37/0xe0 [xfs]
[ 3143.894288] [<ffffffffa038fd39>] xfs_log_reserve+0xb9/0x220 [xfs]
[ 3143.894299] [<ffffffffa0389e35>] xfs_trans Reserve+0x2b5/0x2f0 [xfs]
[ 3143.894311] [<ffffffffa037ad9f>] xfs_setattr_nonsize+0x19f/0x610 [xfs]
[ 3143.894328] [<ffffffffa037b63d>] xfs_vn_setattr+0xb2/0x80 [xfs]
[ 3143.894334] [<ffffffff811d9581>] notify_change+0x231/0x380
[ 3143.894337] [<ffffffff811ed97b>] utimes_common+0xcb/0x1b0
[ 3143.894339] [<ffffffff811ed1b>] do_utimes+0xbd/0x160
[ 3143.894340] [<ffffffff811edc2f>] SyS_utime+0x6f/0xa0
[ 3143.894343] [<ffffffff816a49d2>] system_call_fastpath+0x16/0x1b
[ 3143.894369]
[ 3143.894380] [ BUG: lock held when returning to user space! ]
[ 3143.894384] 3.17-generic #1 Tainted: G        W  O
[ 3143.894386] fs-driver-tests/7127 is leaving the kernel with locks still held!
[ 3143.894388] 1 lock held by fs-driver-tests/7127:
[ 3143.894392] #0: (sb_internal){+,+}, at: [<ffffffffa0389a44>] xfs_trans_alloc+0x24/0x40 [xfs]
[ 3143.894410] LFSV: Fatal error was arisen.
XFS bug

528 int
529 xfs_setattr_nonsize(
530     struct xfs_inode    *ip,
531     struct iattr      *iattr,
532     int               flags)
533 {
534     ...
535     tp = xfs_trans_alloc(mp, XFS_TRANS_SETATTR_NOT_SIZE);
536     error = xfs_trans_reserve(tp, &M_RES(mp)->tr_ichange, 0, 0);
537     if (error)
538         goto out_dqrele;
539     xfs_ilock(ip, XFS_ILOCK_EXCL);
540     ...
541     out_trans_cancel:
542         xfs_trans_cancel(tp, 0);
543         xfs_iunlock(ip, XFS_ILOCK_EXCL);
544     out_dqrele:
545         xfs_qm_dqrele(udqp);
546         xfs_qm_dqrele(gdqp);
547     return error;
Conclusions

- Fault handling in file systems is not in good shape
- Research should be continued
- First conclusions:
  - Fault simulation in unit tests is much more cost effective
  - Systematic tests are more cost effective than random ones
Thank you!

Alexey Khoroshilov
khoroshilov@linuxtesting.org
http://linuxtesting.org/

Institute for System Programming of the Russian Academy of Sciences