LINUX PLUMBERS CONFERENCE
OCTOBER 15-17, 2014 DÜSSELDORF, GERMANY

LinuxTesting.org

Systematic Testing of Fault Handling Code in Linux Kernel

Alexey Khoroshilov
Andrey Tsyvarev

ISP RAS
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

```
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

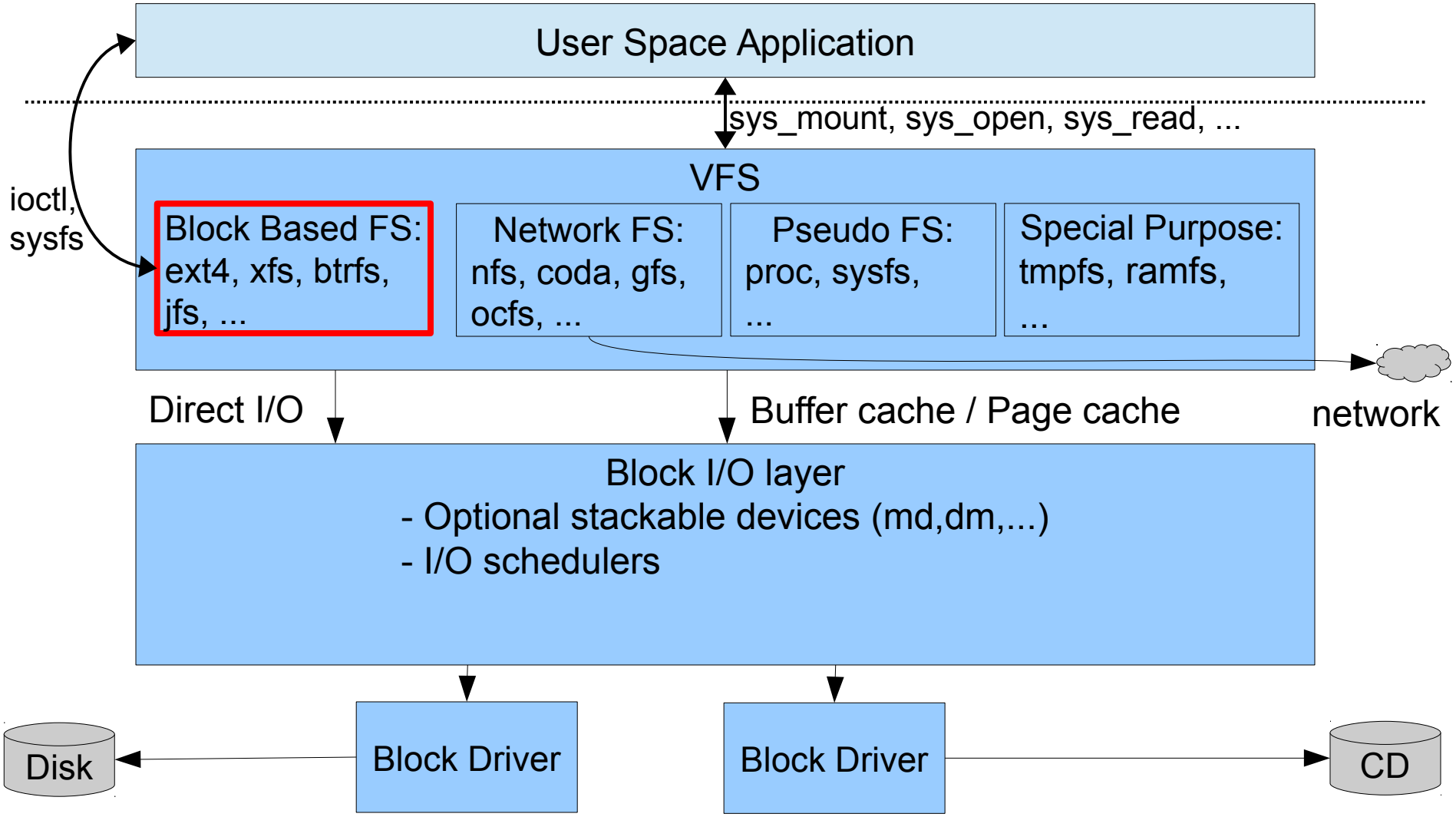
Experiments – Outline

- 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



File System Drivers - Size

File System Driver	Size, LoC
JFS	18 KLOC
Ext4	37 KLoC *with jbd2*
XFS	69 KLoC
BTRFS	82 KLoC
F2FS	12 KLoC

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

File System Driver	ioctl	sysfs
JFS	6	-
Ext4	14	13
XFS	48	-
BTRFS	57	-

FS Driver – Partition Options

File System Driver	mount options	mkfs options
JFS	12	6
Ext4	50	~30
XFS	37	~30
BTRFS	36	8

FS Driver – On-Disk State

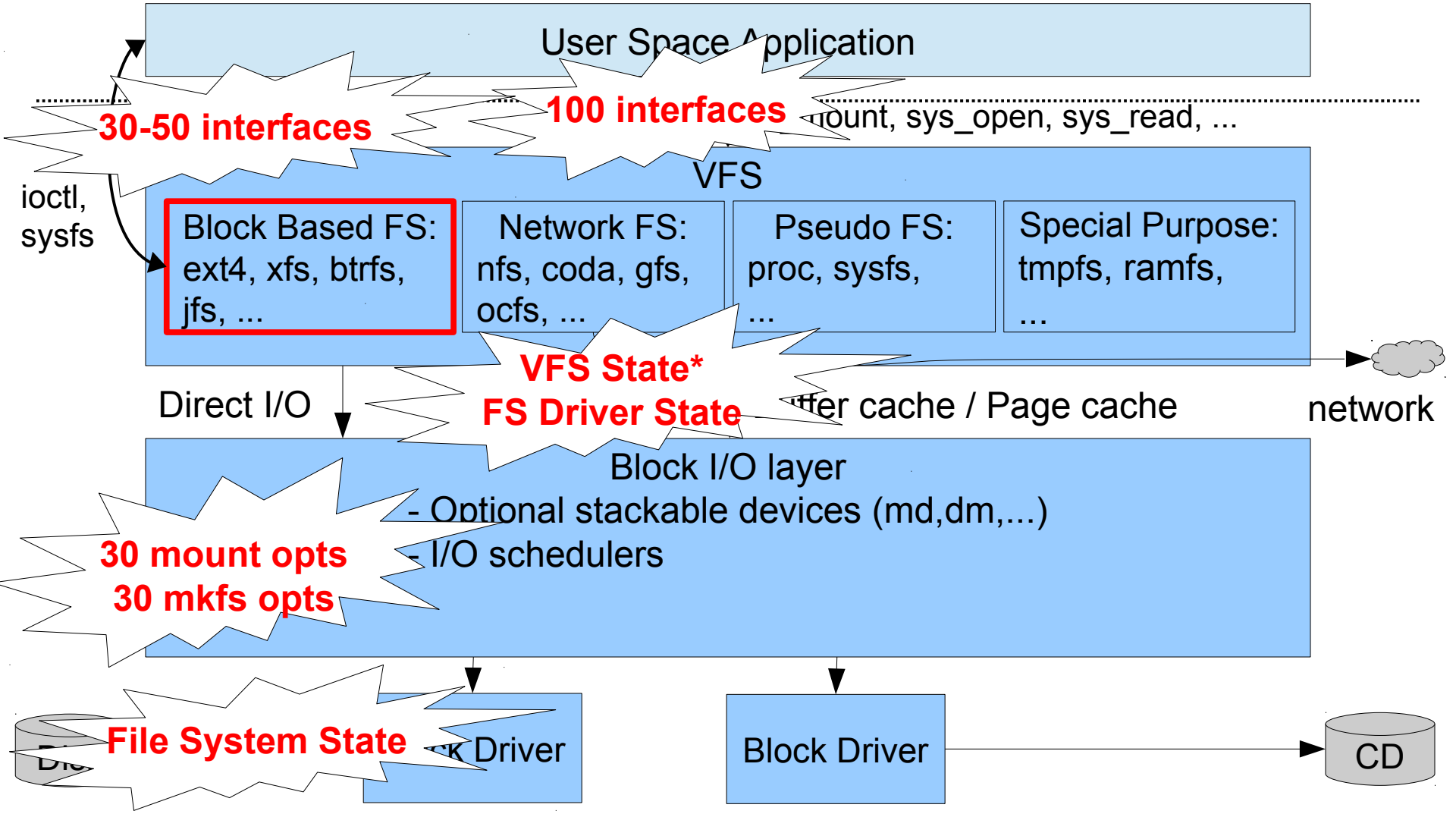
File System Hierarchy

* 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



FS Driver – Fault Handling

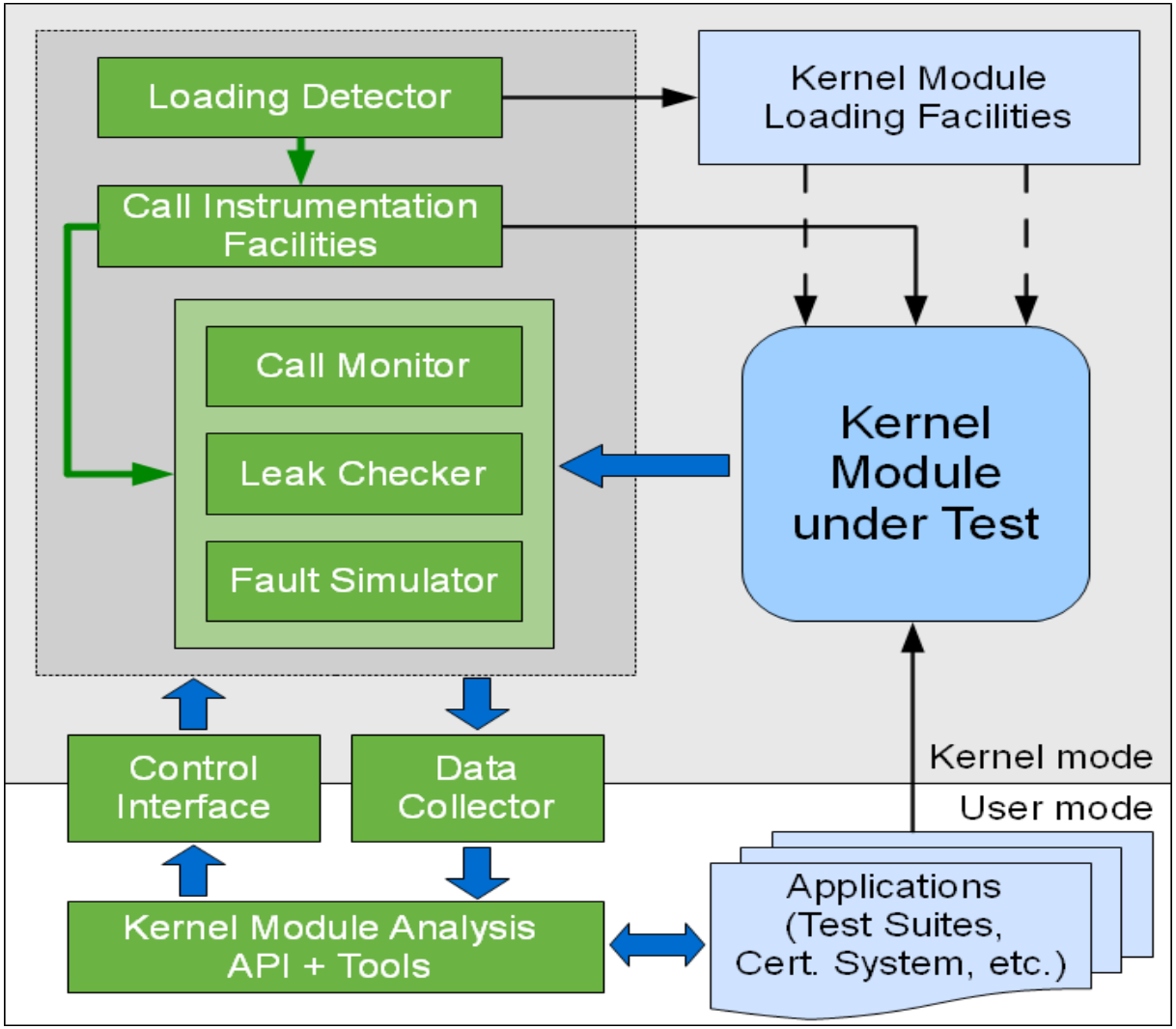
- Memory Allocation Failures
- Disk Space Allocation Failures
- Read/Write Operation Failures

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

	Fault points	Expected Time
Xfstests (10 system tests)	270 327	2,5 years
LFSV (180 unit tests*76 mount options)	488 791	7 months

Possible Idea

- Unit test structure
  - Preamble
  - **Main actions**
  - Checks
  - Postamble
- What if account fault points inside main actions?

Complete Enumeration

	Fault points	Expected Time
Xfstests (10 system tests)	270 327	2,5 years
LFSV (180 unit tests)	488 791	7 months
LFSV (180 unit tests) – main part only	9 226	1,5 hours

- 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

	Increment *new lines*	**Time** *min*	**Cost** *seconds/line*
LFSV without fault simulation	-	110	-
LFSV – main only – no filter	311	92	18
LFSV – main only – stack filter	266	2	0.45
LFSV – whole test – no filter		*unfeasible*	
LFSV – whole test – stack filter	333	4	0.72

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

	Increment *new lines*	**Time** *min*	**Cost** *second/line*
LFSV – whole test – stack filter	333	4	0.72
LFSV – whole test – stackset filter	354	9	1.53
Xfstests – stack filter	423	90	13
Xfstests – stackset filter	451	237	31

+ 10-30 times more cost effective

+ Better coverage

Systematic vs. Random

	Increment *new lines*	**Time** *min*	**Cost** *second/line*
Xfstests without fault simulation	-	2	-
Xfstests+random(p=0.01,repeat=200)	380	152	24
Xfstests+random(p=0.02,repeat=200)	373	116	19
Xfstests+random(p=0.05,repeat=200)	312	82	16
Xfstests+random(p=0.01,repeat=400)	451	350	47
Xfstests+stack filter	423	90	13
Xfstests+stackset filter	451	237	31

Systematic vs. Random

+ 2 times more cost effective
+ Repeatable results
– Requires more complex engine
+ Cover double faults
– Unpredictable
– Nondeterministic

No.	Type	Brief	Added on	Accepted	Status
F0010	Crash	f2fs: Possible use-after-free when umount filesystem	2014-09-08	https://lkml.org/lkml/2014/7/21/198 commit	Fixed in kernel 3.17-rc1
F0009	Crash	ext4: Destruction of ext4_groupinfo_caches during one mount causes BUG_ON for other mounted ext4 filesystems	2014-05-12	https://lkml.org/lkml/2014/5/12/147 commit	Fixed in kernel 3.16-rc1
F0008	Crash	f2fs: BUG_ON() is triggered in recover_inode_page() when mount valid f2fs filesystem	2014-04-18	https://lkml.org/lkml/2014/4/14/189 commit	Fixed in kernel 3.17-rc1
F0007	Crash	f2fs: f2fs unmount hangs if f2fs_init_acl() fails during mkdir syscall	2014-02-17	https://lkml.org/lkml/2014/2/6/18 commit	Fixed in kernel 3.15-rc1
F0006	Deadlock	f2fs: a deadlock in mkdir if ACL is enabled	2013-10-28	https://lkml.org/lkml/2013/10/26/163 commit	Fixed in kernel 3.12-rc3
F0005	Crash	ext4: system hangs after failure in ext4_mb_new_preallocation()	2013-07-01	https://lkml.org/lkml/2013/5/5/64 commit	Fixed in kernel 3.10-rc3
F0004	Deadlock	ext4: deadlocks after allocation failure in ext4_init_io_end()	2013-06-04	https://lkml.org/lkml/2013/5/13/426 commit	Fixed in kernel 3.10-rc3
F0003	Crash	jfs: Several bugs in jfs_freeze() and jfs_unfreeze()	2013-05-24	https://lkml.org/lkml/2013/5/24/76 commit	Fixed in kernel 3.10-rc3
F0002	Crash	ext4: NULL dereference in ext4_calculate_overhead()	2012-11-28	https://lkml.org/lkml/2012/11/28/354 commit	Fixed in kernel 3.8-rc1
F0001	Crash	ext4: NULL pointer dereference in mount_fs() because of ext4_fill_super() wrongly reports success	2012-11-08	https://bugzilla.kernel.org/show_bug.cgi?id=48431 commit	Fixed in kernel 3.8-rc1

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 0000000000008250 ffff8800701e7c18 ffffffffa023f610
[ 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_fsim_cmm]
[ 3143.894230]  [<ffffffffa02401d3>] kedr_intermediate_func_kmem_cache_alloc+0x73/0xd0 [kedr_fsim_cmm]
[ 3143.894264]  [<ffffffffa038b497>] kmem_zone_alloc+0x77/0x100 [xfs]
[ 3143.894277]  [<ffffffffa038fbd7>] 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+0x2d/0x80 [xfs]
[ 3143.894334]  [<ffffffff811d9581>] notify_change+0x231/0x380
[ 3143.894337]  [<ffffffff811ed97b>] utimes_common+0xcb/0x1b0
[ 3143.894339]  [<ffffffff811edb1d>] do_utimes+0xbd/0x160
[ 3143.894340]  [<ffffffff811edc2f>] SyS_utime+0x6f/0xa0
[ 3143.894343]  [<ffffffff816a49d2>] system_call_fastpath+0x16/0x1b
[ 3143.894369]
[ 3143.894380] =============================================
[ 3143.894382] [ BUG: lock held when returning to user space! ]
[ 3143.894384] 3.17-generic #1 Tainted: G        W  O
[ 3143.894386] ---------------------------------------------
[ 3143.894388] fs-driver-tests/7127 is leaving the kernel with locks still held!
[ 3143.894390] 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 {
…
600         tp = xfs_trans_alloc(mp, XFS_TRANS_SETATTR_NOT_SIZE);
601         error = xfs_trans_reserve(tp, &M_RES(mp)->tr_ichange, 0, 0);
602         if (error)
603                 goto out_dqrele;
604
605         xfs_ilock(ip, XFS_ILOCK_EXCL);
…
723 out_trans_cancel:
724         xfs_trans_cancel(tp, 0);
725         xfs_iunlock(ip, XFS_ILOCK_EXCL);
726 out_dqrele:
727         xfs_qm_dqrele(udqp);
728         xfs_qm_dqrele(gdqp);
729         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

LinuxTesting.org

Thank you!

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

ISP RAS
Institute for System Programming of the Russian Academy of Sciences