Using Model Checking to Find Serious File System Errors

StanFord Computer Systems Laboratory and Microsft Research.
Published in 2004

Presented by Chervet Benjamin
Dec 2008 : System Modeling and Analysis

Contents:

Introduction -> Importance of checking a File Systems with model System.

Dynamics that make interesting to check a FS with using Model Checking.

Very Serious Error: Can destroy data or corrupt them irrevocably.

Exponential number of test cases: Solve with model checking state-reduction techniques.

Simple notion about a Modern FS (ext3, JFS, Reiserfs, NTFS…)

- All the operations are Journalized.
  - i.e: all the metadata about modification on the FS are written down and stored in a Journal.
  - Enable to recover after a crash/ power failures.
- The data and operation are buffered
  - The data are firstly written in the cache.
  - Then the cache is flushed.
- Fsck: **f**ile **s**ystem **c**hec**k** : tools that detect the error when the system reboot.

Example of Usefullness of a Journal

- Two operations to delete a file on Linux :
  - 1 - Removing its directory entry
  - 2 - Marking space free for the file and the Inode empty.
- If a crash occurs during step 1- and 2-
  - Orphaned Inode (not free, but not linked to a file).
- Whith journalization, at the reboot, the system will be able to know that it attempted to delete a File, but did not success entirely.
- Without it, need to check every single inode.

Specificities of this paper

- Build a software to model check File Systems. FiSC.
- FiSC is based on CMC which lets run an entire operating system inside a model checker.
- The file systems checked are all journalized and are :
  - JFS: Journalized File System
  - ext3 : based upon ext2 but journalized
  - ReiserFS : from his author Hans Reiser
- This Paper focus on the error that occurs after a reboot while fsck is running.

Overview of FiSC (1) : Elements

Main elements of the Systems:

-CMC, running the Linux Kernel
-File System test drivers
-a permutation checker
-fsck recovery checker.

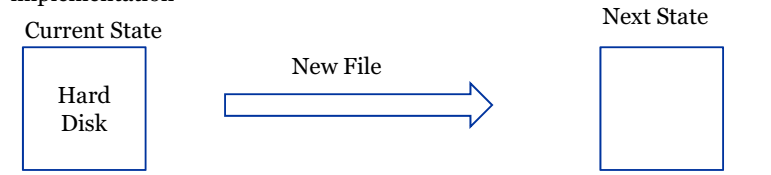
Does not run on a regular File System, but on a simulation of it, throughVirtual File System Interface.

Overview of FiSC (2) : Algorithm

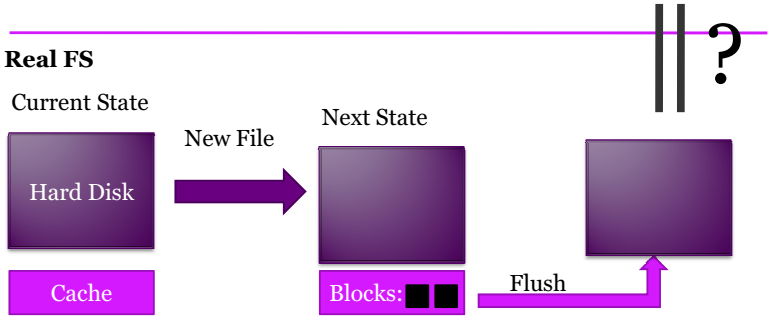
1) CMC start and explore every possible state executions.
2) The test driver do the basic operations (write, mkdir…)
3) At each new state, all the disk writes are forwarded to the permutation checker which check that the Disk is repairable with FsCK
4) We check that the rebuild Disk is conform to his previous Stable Version. An extracted copy of the state is checked.
5) At each step the currently running file system, is compared to a supposed correct FS. This FS reflect the operations done sequentially (not with crash, multi thread…)
6) If and error is detected then the state is marked and the exploration stop.

Correct FS Operation

Abstract FS : very basic implementation



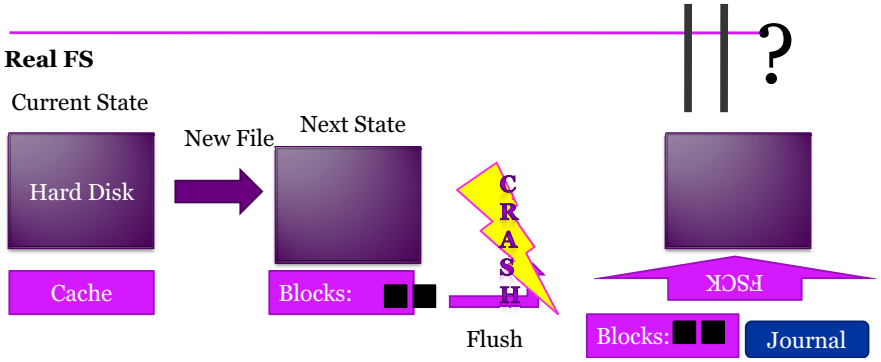
Real FS



Recoverable Disk by FsCK.

Abstract FS : very basic implementation

Real FS



Checking the systems exhaustively but faster.

Use small requirements: ex: 2MB of disk and 16 pages of Memory for ext3.
Downscale everything, reduce the number of nodes, of interactions...

Canonicalization: use the maximum amount of constant values, and reduce the size of data written.

Detect and expose choice points: Transform a non-determinism state (based on time, environment) in a determinism state to expand the number of checked states.

Checks performed by FiSC.

-Deadlock

-Paired functions: Some function for innode allocation need always to be called in paires.

-Memory leak: At each stacks, FiSC look it there is some non referenced Inode.

-The Kernel should not request a ressource, it will not use.

-Make sure that a successful modification of the File System, is user-visible.

-Buffer consistency: Coherence between the journal, and the metadata in the buffer.

-Double fsck:Fsck offer differents running modes. We make sure, that differents Fsck checks provides the same results.

Optimizations

In order to reduce the amount of memory needed in the state model, FISC use severals methods:

-State Hashing and stores states, without much differences on unimportant parameters as a same state.

-Discard the too-large state, given a threshold.

-Search preferentially the state with the most work done to recover from a crash, and thus the most likely to have a problems.

- Stop at the first failures. Not very usefull to explore a state after a failures.

-Stores the simulated Disk space into a database with a hash table. Thus reducing the amount of data need for a state.

-Considers that the FSCK tools do the same task based on the same input disk image.

-Fisc can be run on a clusted, thus making the calculus distributed.

Results :

-32 bugs founds

Error type	VFS	ext2	ext3	JFS	ReiserFS	total
Lost stable data	n/a	n/a	1	8	1	10
False clean	n/a	n/a	1	1		2
Security holes		2	2 (minor)	1		5
Kernel crashes	1			10	1	12
Other (serious)	1		1	1		3
Total	2	2	5	21	2	32

A lot of bugs have been found in JFS, due to the fact that the JFS team was very fast at correcting the bugs, enabling thus FiSC to check for more bugs.

Exemples of bug founds :

```
// e2fsprogs-1.34/e2fsck/jfs_user.h

// Error: empty macro, does not sync data!
#define fsync_no_super(dev) do {} while(0)

// e2fsprogs-1.34/e2fsck/journal.c
static errcode_t recover_ext3_journal(e2fsck_t ctx) {
        journal_t *journal;
        int retval;

        journal_init_revoke_caches();
        retval = e2fsck_get_journal(ctx, &journal);
        /* ... */
        retval = --journal_recover(journal);
        /* ... */

        // Flushes empty journal.
        e2fsck_journal_release(ctx, journal, 1, 0);
        return retval;
}

// e2fsprogs-1.34/e2fsck/recovery.c
int journal_recover(journal_t *journal) {
    // process journal records using cached writes.
    err = do_one_pass(journal, &info, PASS_SCAN);
    if (!err)
       err = do_one_pass(journal, &info, PASS_REVOKE);
    if (!err)
       // writes persistent data recorded in
       // journal using cached write calls.
       err = do_one_pass(journal,&info,PASS_REPLAY);

    /* ... */

    // Write all modified data back before clearing journal.
    fsync_no_super(journal->j_fs_dev);
    return err;
}
```

During the recovering, the journal is erased and then the data are flushed.

The problem come from the Fsynch_no_Super function which is not defined.

Thus, the journal can be cleared before the recovery is really effective.

Others kind of bugs founds :

-Buggy transaction abort at recovery.

-Data loss due to wrong return code number, or inopportune sector rendering.

-Kernel crashes

-Memory leak

-Security hole: -Disk overflow, by memory leak.
-Gain the access to unauthorized files.

Lessons lernead from Fisc:

-Model checking can help the development of Kernel System.

-False positive:
- -Most of the time bug in the model checking harness or in the Understanding of the modeled File System.
- - Developer intentionally violate the properties temporally.

-Limits :
- - No tested for big systems (huge number of files, memory…)
- - No Multi Threaded
- -No low level errors detection
- -Some garantees (disk quota, access control) are not checked
- -Some states are missed (due to discarding).

One of the first application of model checking more complex that simple program checking. Will maybe lead the way for same effectiveness in others domains.