Effective Power Interruption Testing
How to Fail Best

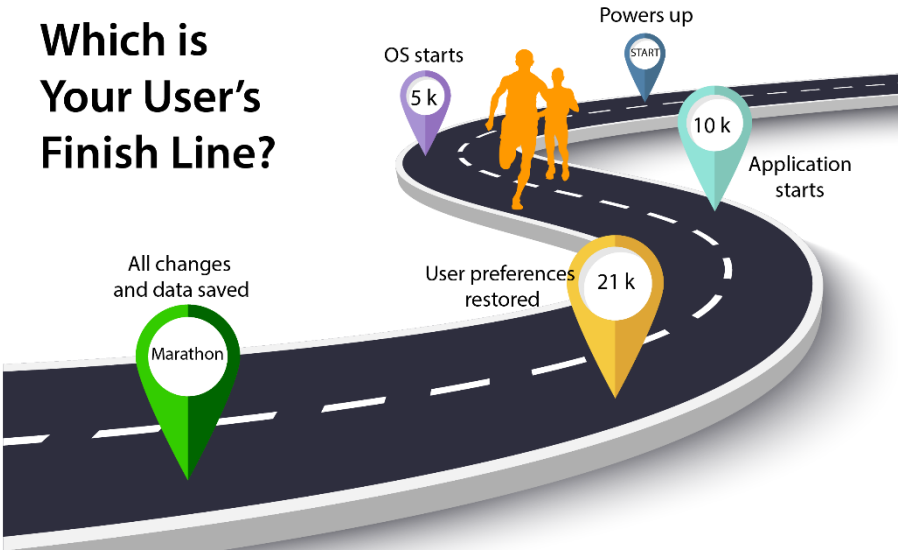
Thom Denholm, Datalight Inc.

From dropped batteries to system failures, embedded designs require solid power interruption testing. Durability demands for embedded products have increased as the expected lifetime of high reliability products has grown. Faced with these conditions, developers must expand their testing toolkit. To achieve the most comprehensive reliability testing in the least amount of time, stress testing must utilize I/O at the point of power interruption.

When your team is responsible for validating the reliability of a design for the embedded marketplace, you need to do more than just tick a box or read a marketing document. Real testing of reliability involves getting into the guts of the embedded design, and covers everything between the application and the hardware. This whitepaper is focused on testing the file system and its interactions. Catching a vulnerability in design or testing is far more efficient (and far less expensive) than handling problems in the field.

Definition of Reliability

Reliability can be defined as: whatever a given customer thinks it is. Powering up and starting the OS are a requirement for nearly all devices, and being able to start the primary device application is very important. Without those, the user must wait for a system restore. Many customers expect their favorite settings or programmed routes to be available and reflected in their next use of the device. Elite customers (and some certifications) require an even higher definition of reliability. They need to start up with settings changed immediately before the power was lost.



It is safe to say that even though the needs of the individual customer vary, the testing bar must be set quite high. Likely higher than the average developer expects.

System reliability vs Data on the media

Examining the previous cases from the computer point of view, the first is related directly to the system files. Sudden power loss could corrupt other files, as long as it doesn't touch the system files. One trick used by developers is to put the system files on a separate partition or even separate media to achieve this goal. Files can be marked read only, though that only keeps the file system from touching them.

To save the end user settings, those changes have to make it to the media. On embedded Linux, for instance, those changes must make it through the write-back cache and any buffers. The data also needs to clear any hardware block cache. In time, all the data will eventually be flushed – if the user waits long enough, their settings will be saved.[1]

In addition to user data, the file system metadata also must be in the appropriate place. For most file systems, Linux flush and fsync commands are used to achieve complete control of the file system, which is the only way to ensure data is committed to the media.

That control over the file system and block device are the key to the final use case, where the data must be committed immediately – waiting for the block cache to time out is not an option.

The next step is to refine the granularity even more – what happens when the power is interrupted?

Write interrupted mid-block

Writes can be interrupted at any point, primarily when power is lost. This results in two options – the system could be in the middle of writing a block or in between writing blocks.

On older magnetic media, an interruption in the middle of writing a block would leave a partial write. The block being written is corrupted – it likely contains fragments of new data and old data. If this was the only copy of that data, it is lost – and the file that contained it is probably useless.

New NAND based media, from SSD and eMMC to SD and CF, have an even larger problem with interrupted writes. It is so big that vendors specifically recommend maintaining power during block writes just to avoid it. Assuming that media power is maintained with a capacitor or other technique, an interrupted write becomes instead write interruption between blocks.

Write interrupted between blocks

From the user perspective, each write consists of one or more blocks of data, along with any metadata that must be written. If a given write is a small amount of data, up to one block, with no metadata changes, then power interruption won't be a problem. For larger writes, an interruption between write blocks will interrupt what is known as an atomic operation.

[1] https://www.datalight.com/resources/whitepapers/reliably-committing-data-in-linux

File systems such as FAT don't do anything special for an atomic operation, so interrupting one of those is not much better than a mid-block interruption on magnetic media. The file being written to has some updated blocks and some not yet updated. For most user data, this is enough to render the file useless.

Metadata should also be part of an atomic write. If it isn't, one of two situations can occur. If the data is written but the metadata has not been, the data would be lost as the system recovered from power interruption. A potentially worse case is if the metadata is written but the data has not yet been. A system recovering in this situation would then try to open this updated file and find either garbage or, worse, other data from the media – a potential security risk.[2]

Testing to make sure longer writes are committed the way a system designer expects is another requirement for comprehensive testing.

Validate the written data

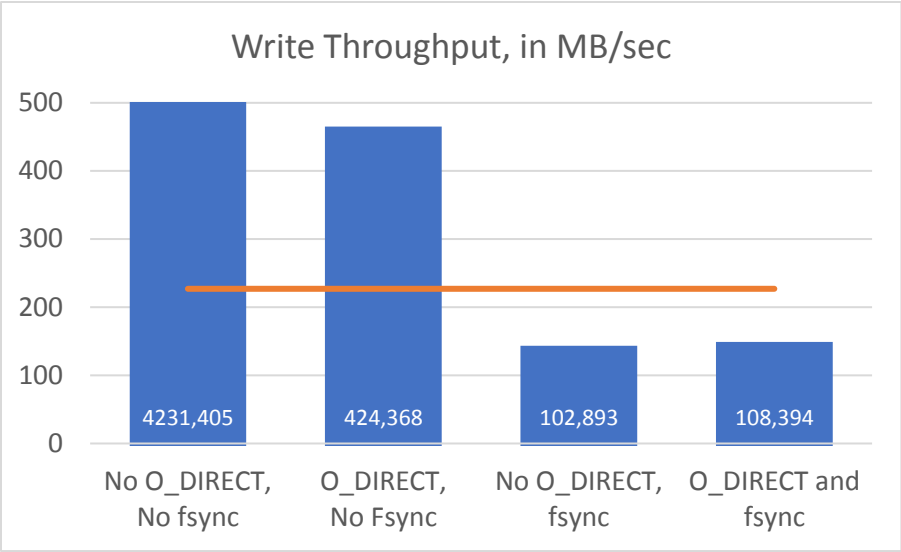
At this point, the developer has an expectation of how data will be written and/or recovered as part of a power interruption. This is all assuming the writes happen in the order they were initiated, which is not always the case. Complete testing should validate the expected state of the media after interrupted writes.

All I need is O_DIRECT

On Linux file systems, there is a particularly persistent myth that opening a file with O_DIRECT is all that is required for data to be reliability stored on the media, and the fsync() command would not be necessary in this case.

We measured the performance of sequential writes using fsync, O_DIRECT, and neither. The orange bar represents the maximum speed of the media, 225 MB/sec. This demonstrates that O_DIRECT is writing only to the DRAM cache, and is not sufficient to guarantee reliability.

[2] http://www.linux-magazine.com/Online/News/Linus-Torvalds-Upset-over-Ext3-and-Ext4



Stress!
Flushes, user and metadata writes, and handling interruptions during and between block writes covers the basic reliability testing for normal system use. The next thing to focus on is system stress – unusual cases that will likely occur in the field.

The first of these that most developers don't examine is what happens when the disk is full. Besides the potential performance implications, a disk full situation can generate more writes as garbage is collected. Related to that are extreme situations in the number of files – does the system latency increase noticeably beyond the first 100 or so files? Both of these situations lead to a larger write error window – and a larger potential for failure.

Another stressor is a system update. This is a situation that is especially important to test thoroughly, especially when power interruption may occur. Atomic writes here can be a major factor which allows the device to recover from a failed update.
Extreme use cases also hit the media especially hard. In the case of NAND flash media, thousands of reads from a given file can cause read disturb, adding bit errors to the media. If these bit errors are not taken care of (a process called scrubbing), the correctible errors can grow to uncorrectable errors.
Other potential media failures include some of these items:

- Specific write patterns – on NAND media without randomization, writing all zeroes (0x00) is actually worse for the flash than writing other patterns.
- Hot Spots – media locations that are prone to failure for reasons unknown to the developer (and possibly even to the vendor)

Discards (the Trim statement)
Another storage related item is discards – using the trim command to inform the media that data is no longer in use. On devices where discards are not used (or under-utilized), latency can increase noticeably once all the blocks on the media have been written once.

This increase latency causes a noticeable drop in performance; when writes take longer, the potential error window grows.

For that matter, the firmware on most NAND based solutions is a black box. What is happening when the media is busy discarding data or wear leveling or garbage collecting? What happens when the power is interrupted during those operations? Using the file system to generate these sorts of failures can be very hit or miss, and a custom media test should be developed to validate its operation during a variety of power interruptions.

Tactics for effective testing

We have examined a number of methods that can be used to generate more effective power interruption testing, so now we must put them all together.

First of all, interrupting an embedded device while quiescent or while reading will result in nothing being lost, unless it is updating something in the background. So power interruption testing needs to trigger those background writes where possible and, most importantly, focus on the writes. Random power interruptions during writes are most likely to exercise the safety routines of the file system and application.

The next step is validating the data written, not just the structure. Make sure that what is most important to your customer is being confirmed here – overwrites and completely atomic operations. If data is overwritten in place, then an entire operation must be atomic to prevent a corrupt state – half old data, half new data, all useless.

While the user data is important, the system files can be even more important. If corruption of other files affects system files, the entire device can be rendered unusable. The same could happen if power interrupts an unprotected system update.

Check the disk

Most file systems provide a utility based on chkdsk to detect these sorts of failures, though they can't usually correct them very well.

The original MS DOS chkdsk found data that was on the media but not represented in the file allocation table (FAT) and also could find a number of errors within the FAT. It was not able to connect that data to file names, so files full of lost chains and other errors resulted in nearly useless files such as FILE0001.CHK – taking up space but not useful for most applications.

Conclusion

The best testing meets both the needs of the strictest users and the goals of stressing the system in the best way. Interruptions during writes will demonstrate the most failures and accurately reflect field stress. Other factors to consider include validation of data, atomic operations and cases that also stress the media. Real testing of reliability is more than just a requirement, it also leads to long term success of the embedded design.

Author

Thom Denholm is an experienced embedded software engineer, combining a strong focus on operating system and file system internals with a knowledge of modern flash devices. He has a degree in Mathematics and Computer Science from Gonzaga University. His love for solving difficult technical problems has served him well in his seventeen years with Datalight. In his spare time, he works as a professional baseball umpire.