Technologic Systems
16525 East Laser Drive
Fountain Hills, AZ 85268
Phone (480) 837-5200

Preventing Filesystem Corruption in Embedded Linux

Introduction

A common concern with embedded Linux design is the risk of filesystem corruption when power to the system is unexpectedly shut off. It is our belief that many undiagnosed embedded system failures have power failures as a root cause. System designers can deal with this in a variety of ways, and the solution is different for different applications. This paper presents some steps you can take to make your system perform robustly with minimal maintenance.

Motivation

Almost any computer system is subject to unexpected power failures. For some embedded systems, this only occurs when the power grid goes down. For others, it may happen when a user decides to pull the plug instead of using a documented shutdown procedure. Automotive and nautical systems need to anticipate that power will stop and start several times a day.

If an embedded system is implemented without thinking about what happens when the power goes down, it may run for weeks, months, or years before users experience an unexpected and catastrophic failure. From the user's perspective, their device worked fine yesterday and today it doesn't even turn on. A costly debugging process reveals what happened. A file that was crucial to Linux startup, such as */sbin/init* or */lib/libc.so*, has been corrupted.

Other symptoms of filesystem corruption can be more insidious, such as missing configuration files for your application. Bug reports from a system that is getting random corruption will come in all shapes and sizes, and can take many man-weeks to debug.

Explanation

It may not be immediately obvious that files such as */sbin/init*, which never get written, and which may even be marked in the filesystem as read-only, can still become corrupted. Dissecting the inner workings of filesystems and the Linux block layer is beyond the scope of this paper. It suffices to understand that the filesystem will issue writes in a minimum size, typically 4 KB, and a single 4 KB block may have data in it that is part of two different files. There is nothing that says those two files need to be in the same directory, or have the same access permissions. Thus, a simple write to a log file can result in a read and rewrite of part of any file on the partition. When power goes down in the middle of that rewrite, the result is silent data corruption.

Filesystems also have to modify a lot of metadata in various places in order to just create a one byte file. A power failure during that operation could, for example, destroy the names of several other files.

These issues are a concern with a magnetic hard drive, but a flash based storage system adds further layers of complexity. NAND flash chips are divided into erase-blocks of typically 128 KB or 256 KB. Writes at a low level happen one erase-block at a time. Thus, a write interruption could corrupt 128KB of data. Flash specific filesystems attempt to avoid this problem by doing their writing in a journaling fashion. The Technologic Systems XNAND driver performs atomic block writes as a side effect of its multiple redundancy. SD cards and compact flash cards have their own proprietary hardware algorithms which may make no guarantee about block write atomicity.

Solutions

Use a Read-Only Root Filesystem

Files that don't need to be modified should be kept on a partition that is mounted read-only. The simplest solution to avoiding power failures during writes is to avoid all writes. Mounting a partition read-only carries a lot more weight than file permissions. In most cases it means that there will be no modifications to the entire partition*, and as a result there is no risk of corruption.

The fastboot environment provided by default on all recent Technologic Systems SBCs serves as an example of how Linux can be run with everything read-only. It uses a simple initial ramdisk for the root filesystem, and mounts a more complete Linux filesystem read-only so that the binaries are accessible. For many embedded systems, this setup is entirely adequate. Data logging, if needed, can be accomplished over the network.

If local data logging is required, a read-write partition can be established for that purpose only. With this setup, the worst-case scenario with a poorly timed failure is that the system will boot correctly, but the data it has been collecting recently will be corrupted.

If the sophisticated features of a distribution like Debian are required, it is usually assumed that the root filesystem is read-write. Our recent products offer another alternative which uses unionfs to run Debian so that all filesystem changes exist only in RAM. A read-write partition can then be established only for data-logging. This allows for a full-blown system running a LAMP website with a minimum amount of risk. This page explains how to set up a TS-73xx SBC in the same way.

Use the Right Filesystem

Linux offers many filesystems, but we have been able to expose bugs in most of them in at least some kernel versions. The only one we have complete confidence in is ext3. When in doubt, ext3 should be used for any partition that will be mounted read-write.

The ext2 filesystem is much more fragile. It is a perfectly good filesystem for systems that are able to mount it read-only, or unmount it correctly. But our experience shows that corruption is extremely likely with a power failure on ext2.

Recent Posts

- Preventing Filesystem Corruption in Embedded Linux
- How to Write an SD Card Image (Linux, Windows, Mac OSX)
- Getting Current Voltage Input (VIN) on TS-7670 or TS-7400-V2
- Develop a Simple Qt Quick Interface for HMI/SCADA Applications
- Reading CPU Temperature and Controlling LED with C++ via sysfs

Blog Search

Search for:
Search

Newsletter Subscription

Sign up for monthly newsletters to get updates from our blog, product announcements, industry highlights, and more.

Name
Company
Email address:
Your email address
Sign up

Contact Us

Your Name (required)
Your Email (required)
Subject
Your Message

The other option we recommend considering is jfs. We have found that the jfs filesystem is not reliable in Linux 2.6.24, however in 2.6.21 it is extremely robust. We recommend jfs on the TS-7800 and TS-73×0 products. Our anecdotal evidence suggests that corruption is less likely with jfs than with ext3. Jfs is also has a quick mount time and file system check time.

Build an Emergency Backup into Your System

In many cases, the failure of an embedded system in a remote location costs many thousands of dollars just to reach it and replace it. If this is true of your system, it is better to make an investment up front to prevent failures.

Many Technologic Systems SBCs have a small amount of non-volatile memory available in the RTC module, the on-board microcontroller, or an NVRAM chip. If none of these are available on the model you are using, it may even be worthwhile to designate an entire partition for this purpose, or to store this data remotely over a network, even though you only need one bit of data. Lacking any of those options, you can use the root filesystem for this purpose. This storage space, along with the watchdog timer, can be leveraged to make sure a Debian initialization is successful, and do something about it if not.

Most Technologic Systems ARM SBCs boot to an initial ramdisk. The kernel and initial ramdisk are stored on separate raw data partitions, so they are not subject to filesystem corruption. A failure during this initial boot process is only possible in the event of a low level hardware failure. Such a failure is even less likely on boards that boot from XNAND.

This paper is concerned with what happens after the initial boot. At this point a typical Debian system proceeds to mount a Debian root partition and boot to it. If the Debian partition is corrupted, this is where the problem is. Our fail-safe SD card image for the TS-7800 demonstrates a way to guarantee a successful boot process. It uses the non-volatile RAM in the RTC to record whether a restore process is necessary. The boot procedure is as follows:

1. Linux boots to an initial ramdisk.
2. The RTC value is checked to confirm a normal boot sequence.
3. A value is written to the RTC indicating failure, the watchdog is started, and Debian is booted.
4. The watchdog is continually fed during Debian initialization.
5. The RTC value is written to indicate success, and the watchdog is disabled.

In the field, step 5 is to be replaced by the application software. The application can be responsible for feeding or disabling the watchdog, and for writing different values to the RTC.

If a failure code is read at step 2, the entire Debian partition is reformatted and restored based on a read-only backup partition. This configuration results in a robust system that is always able to boot Debian. The worst-case behavior is loss of logged data and a lengthy delay while the system is restored.

Build a Battery Backup into Your System

Sometimes, the cheapest way to make your system reliable is to simply make sure power never fails. For low quantity/high reliability embedded applications, the extra cost for an uninterruptible power supply can be a very good investment.

TS-BAT3, TS-BAT10, and TS-BAT12

The TS-BAT3 and TS-BAT10 are PC/104 power peripherals that can supply 1000 mAh or 2000 mAh, respectively, of 5V power to an embedded SBC when external power is not available. They communicate to the SBC using a serial port, so the SBC can intelligently shut down if necessary. For more power intensive applications, the TS-BAT12 is a stand alone, fully enclosed battery backup solution providing 5 Ah of 24 V power (more than 4 hours under 500 mA load, or 11 W). The TS-BAT3, TS-BAT10, and TS-BAT12 are strong solutions for an embedded device that needs to shut down gracefully in all cases.

TS-SILO

Offered as an on-board, soldered solution in our newer single board computer products like the TS-7680 and TS-7553-V2, TS-SILO uses super capacitors to provide 20 – 60 seconds of power hold. This is enough time to gracefully shutdown when a power outage is detected. Once depleted of stored energy, TS-SILO can be fully recharged in under a minute. You can read more about TS-SILO in our TS-SILO press release.

Conclusion

Filesystem corruption is a frequent problem for embedded Linux systems. System designers need to make a business decision regarding how much per-unit cost and how much engineering to put into preventing it. The inputs to this decision are different for every application. Designers need to weigh the business costs of failure prevention against the business costs of occasional failures in the field. For some systems, it may be enough to provide the customer with an extra SD card that they can install in the event of a failure. For others, a failure in the field is prohibitively expensive, and it is best to spend a significant amount of money per unit to prevent it.

This paper outlines four strategies that can reduce or eliminate failures due to filesystem corruption:

- Mounting filesystems read-only
- Choosing the best filesystem
- Using a software backup
- Using a battery backup
- Using a TS-SILO equipped solution

The success of an embedded product may depend on evaluating these options during the design phase.

* This is not always true for NAND flash. Data stored on NAND flash can be corrupted after too many reads or after the passage of years. As a result, flash-specific filesystems such as YAFFS2 may move data around at times even on partitions that are read-only. Technologic Systems XNAND driver, unlike flash-specific filesystems, implements the NAND driver in an entirely separate software layer from the filesystem. The filesystem can be mounted read-only and the NAND driver will still rewrite flash blocks if necessary to maintain data integrity. A power failure during that extremely rare operation is still not likely to cause data corruption.

Similar Posts

Archives

July 2016
June 2016
May 2016
April 2016
March 2016
February 2015
January 2015
September 2014

RSS - Posts
RSS - Comments

Powering an Embedded Single Board Computer From a Battery
Reliable In-Vehicle Data Logging and Tracking
How to Write an SD Card Image (Linux, Windows, Mac OSX)
Practical Guide to Getting Started with the TS-7250-V2
Who's (Not) Afraid of the Dark?

Posted July 19, 2016 by Derek Hildreth
Whitepapers

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.