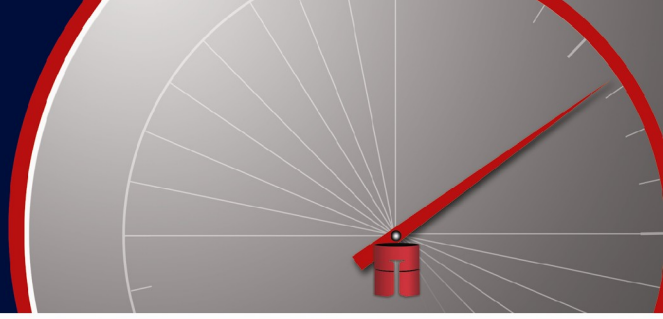

Database and File System on Embedded Devices

Introduction

On embedded devices, like other computer systems, persistent data is saved in files on a file system. If an application crash, system crash or power failure occurs while data is being written, pending changes will be lost. Even worse, the partial changes that are written can leave files in an inconsistent state, resulting in corrupt data or even missing files.

The problem effects both the structure of files and the contents of the file. The file system maintains information about file structure. Applications are responsible for file contents.

There are three ways for a file system or application to cope with crashes:

1. Careful-write: Order writes carefully so that if a failure occurs partway through, the data is left in a consistent state.
2. Lazy-write: Do nothing to prevent data from being left in an inconsistent state, but provide a way to detect and fix errors that occur as a result.
3. Journaling: Write new changes to a staging area without destroying the previous state until a new consistent state is achieved.

To improve performance, many device drivers use a write cache to buffer changes in memory and actually send them to disk as a group, possibly reordered. Since careful-write relies on the order that data is written, it cannot take advantage of a write cache and performance suffers.

Lazy-write requires a lengthy disk checking task when a crash is detected to search for errors. The file system can be repaired, but files or parts of files can sometimes be lost in the process.

Journaling offers the best trade-off between performance and reliability. Changes are written in such a way that the previous state is not destroyed until the write operation is fully completed.

Traditional File Systems

The file allocation table (FAT) file system is the most widely supported file system. Because of its relatively simple and straightforward design, FAT is available on almost every operating system and is the default operating system for portable

storage media.

However, due to its simplicity, FAT suffers performance problems when files become too large and is easily corrupted in case of a system failure.

Linux and similar operating systems need additional features not found in FAT, such as file ownership and permissions, for best operation. The most widely supported operating system on Linux is ext2, which provides good performance and scalability. But like FAT, ext2 can become corrupt after a system failure.

Journaling File Systems

A journaling file system is a type of file system that protects information about files in case of a crash. Some journaling file systems also protect the contents of files, but usually the application has no way to know when the changes it has made to a file will be truly written and recoverable.

TFAT

Windows CE 5.0 introduced a transactional version of FAT called TFAT that protects against corruption and data loss when an interruption occurs. TFAT is not available on any other operating systems.

TFAT uses two copies of the FAT table, a stable copy and a working copy containing pending operations. After a file operation has been completed successfully, the working table is copied to the stable table. This protects the file system, but does not protect the contents of files. The additional overhead decreases performance compared to regular FAT, which is already slow.

TFAT can be configured to protect file contents as well, but only when the write cache is disabled. This reduces performance so greatly that it should rarely be used in practice.

TFAT can be used on removable media that is shared with other operating systems that support FAT, but some

File Structure Errors

Journaling file systems always prevent the following errors when an interruption occurs:

A file is lost when created or moved to a new location.

Blocks of data in a file are missing after the file size is changed.

An unwanted file, such as a lock file, remains on the file system after being deleted.

File Content Errors

Journaling file systems can sometimes be configured to prevent the following errors:

When a series of blocks are written to a file, the last blocks are not written.

When a series of blocks are written to a file, some blocks are not actually written, regardless of the order in which they were written.

operations are unavailable, such as removing directories.

ext3

Just as TFAT extends FAT, ext3 adds a journal to ext2 to protect against data loss. Instead of being a complete copy of the allocation table as in TFAT, the ext3 journal contains only the list of changes made by each operation, which allows it to perform well even with a large number of files. All recent Linux distributions support ext3.

Several levels of protection are available with ext3. Typically, only file system changes are written to the journal, but data changes can be written as well. However, this causes every block of data to be written twice and cannot be applied selectively even when an application would know that the extra copy is unnecessary.

More robust journaling file systems, such as JFS and XFS, are supported by some versions of Linux. These file systems typically have a high amount of overhead for simple operations and are usually not appropriate for embedded devices.

JFFS2

A special-purpose file system for flash media, the Journaling Flash File System 2 (JFFS2) is a log-structured file system. Instead of using a separate journal, all updates are stored in a log format. JFFS2 works on Linux and eCos.

The benefit of this approach is that all writes are transactional and spread across the media, reducing wear on fragile flash media. JFFS2 is not appropriate for magnetic media like hard disks because the contents of each file are scattered throughout the file system, requiring much more physical activity than a traditional file system. Flash media, however, does not have a problem with such random access.

Datalight Reliance™

Reliance™ is a file system that uses transaction points to preserve both file structure and file contents in the event of an unexpected power interruption. Reliance™ can be used on Windows CE™, vxWorks, Nucleus, and others.

Reliance™ always writes new data to free blocks, instead of overwriting existing data. After all data is written, a transaction point is set and the changes are committed permanently. Only then are the old data blocks made available for subsequent write operations. If there is an interruption before the transaction point is set, only the new data is lost and the file remains in a consistent state.

Transaction points can be configured to occur automatically, or can be triggered explicitly by an application. Automatic transaction points occur when certain file operations are performed, such as when closing a file. Timed transaction points are executed periodically by a background thread. Programmable transactions are set by using special IOCTL functions in an application. A transaction point applies to the entire file system, not just a single file.

ITTIA DB™ Embedded Database

ITTIA DB is a database used by applications on embedded systems and devices to efficiently protect data in case of an unexpected interruption. ITTIA DB stores data in a file on a file system, and uses a separate log file to protect the contents of the file.

ITTIA DB relies on the file system to protect the structure of the database and log files, and should always be used with a journaling file system when possible. ITTIA DB does not require the file system to protect file contents, however, and can be used safely and efficiently with TFAT and ext3 in the default configuration.

When ITTIA DB is used with a file system like Reliance™ that protects both file structure and file contents, the log functionality can be safely disabled when the database is opened. The application must set transaction points every time a database transaction is committed. Also, transactions cannot be rolled back in this configuration.

In addition to reliability, ITTIA DB adds features such as indexed search, relational data modeling, and shared access that improve the performance and robustness of an application.

Conclusion

Embedded systems and devices require reliable storage that is self-maintaining even in the case of a system failure or power loss. When a file is lost or becomes corrupt on a desktop computer, it can usually be fixed manually. When a similar problem occurs on a device, the device can stop working completely.

Selecting a reliable file system and database technology can greatly improve the reliability and performance of a device.