Survey of Filesystems for Embedded Linux

Presented by Gene Sally

CELF

Presentation

- Filesystems In Summary
  - What is a filesystem
  - Kernel and User space filesystems
  - Picking a root filesystem
- Filesystem Round-up
  - Slide-by-slide description of filesystems frequently used by embedded Linux engineers
  - NFS and initramfs filesystems

Diversion for Those New to Linux/Embedded: The Kernel and Filesystem

- **The RFS and Kernel are separate entities.**
  - **Related? Yes, but not so tightly bound that they can't change independently.**
- **A filesystem must be present for the kernel to start successfully.**
  - **Can be an in memory filesystem, network filesystem**
  - **Can be "attached" to the kernel image loaded into memory**
  - **This filesystem mounted at /, aptly called the root filesystem (RFS)**
  - **Can have a system with s**
- **The Linux kernel, after sta and execute some progra**
  - **While they may be packag separate entity from the ke**

For those new to using Linux for an embedded project, having a separate kernel and user-space takes some explaining, even for those who use Linux on their desktop.

Filesystems in Linux: General Features

- Linux (like Unix) is designed to use any number of arbitrary filesystems
  - Provides uniform interface to filesystems through the VFS (Virtual FileSystem)
  - Provides shared routines (like caching)
  - Physical storage not necessary (think proc filesystem)
- Filesystems implemented as kernel modules
  - Most of the time (for embedded systems) compiled directly into the kernel
  - Can be loaded as modules after kernel starts
- User space filesystems: FUSE
  - Fully functional filesystems that run in user space
  - Intriguing solution for embedded systems, more stable kernel

Linux Virtual FileSystem

- Around Since Linux 1.0
  - File-oriented nature of *nix OS makes it important to get this right
  - ext/ext2 filesystems used the “emergent” VFS in Linux 1.0
  - As OS matured, more functionality migrated to VFS layer, with ext2 often serving as the model and test case
- Housekeeping
  - Registration, removal
  - Enumeration (cat /proc/fs)
  - Associate physical devices to filesystem drivers
  - Synchronization
- Common Code
  - Node handling
  - Look-ups
  - Caching

FUSE Filesystems

- **Part of the kernel starting at 2.6.14**
  - **Kernel module**
  - **User land helper programs and library**
  - **Patches for 2.4.21**
- **Sample Filesystems**
  - **Media: DVD, Playlists, MythTV**
  - **Dynamic Devices: USB**
  - **Interesting: Database, Encrypted, GMail**
- **Language Bindings**
  - **C, C++, Java, C#, Python,**
- **More Info**
  - **http://fuse.sourceforge.ne**
  - **http://fuse.sourceforge.ne**

Not very space efficient or high-performance in its current release, so not super-useful for embedded applications. But keep your eyes peeled!

VFS "Traditional" Filesystems

- Implemented as filesystem drivers that plug into the Linux VFS architecture
- Lots of these! For desktop users, the following may be familiar:
  - Ext3, ReiserFS, NTFS
- Embedded Systems typically use specialized filesystems
  - ext2
  - cramfs
  - JFFS2
  - squashfs
  - YAFFS2

Picking an RFS

- **Right for the device**
  - Flash devices require a wear-leveling filesystem if you're using it for read-write.
  - If you're short on space, pick a filesystem that allows you to control block size and that doesn't store complete metadata.
- **Right for the application**
  - Read/write when necessary
  - Read-only filesystems need extra work at boot time to create writable partitions expected by the operating system.
  - Remember – RAM-based filesystems reduce memory available to the kernel or applications.

ext2: Second Extended Filesystem

Description	Ext2 shipped with Linux from the start. Most systems today use the journaling cousin of ext2, named ext3.
When to Use	▪ Ramdisks ▪ Low-resource systems
Capacity and Limitations	2 TB, 10^{18} files Full complement of file ownership and permissions
How to Use	Most systems ship with ext2/3 drivers and utilities as part of the distribution. Typical usage pattern is to create a partition directly on a block device, or use a loopback block device that is bound to a file.
Home Page **More Info**	http://e2fsprogs.sourceforge.net/ext2.html http://lldn.timesys.com/tag/ext2

cramfs

Description	Compressed ROM Filesystem. Read only filesystem widely used in the embedded space. Data stored in compressed format (zlib).
When to Use	▪ Low-memory systems ▪ Ensures RFS integrity ▪ Metadata not important (doesn't store full information)
Capacity and Limitations	256 MB, 2^{16} files Does not store all permissions, all files owned by root. No timestamps stored (inode overhead is just 12 bytes!)
How to Use	`$ mkcramfs –m dev.cramfs.txt <rfs_dir> rootfs.cramfs` Full details at: http://lldn.timesys.com/docs/cramfs
Home Page More Info	http://sourceforge.net/projects/cramfs http://lldn.timesys.com/tag/cramfs

squashfs

Description	Read only filesystem that includes several improvements over cramfs, notably in compression and metadata storage. Adjustable block sizes allow a user to create filesystems that compress better.
When to Use	▪ Low-memory systems ▪ Need control over the endianness
Capacity and Limitations	2^{32} GB, 2^{32} files, Page size from 2^{12} to 2^{18} A files owned by root Read-only
How to Use	`$ mksquashfs $RFS ./squashfs-rfs/rfs -nopad -all-root` The resulting file can then be written directly to a flash partition. Use rootfstype=squashfs on the command line, mounting the /dev/mtdblock device as the root device.
Home Page More Info	http://squashfs.sourceforge.net http://www.artemio.net/projects/linuxdoc/squashfs http://lldn.timesys.com/docs/tiny_flash

romfs

Description	Minimum filesystem, very small kernel module. The “rom” in romfs doesn’t refer to the hardware “ROM”.
When to Use	▪ Trying to make as compact a kernel as possible ▪ Initial RAM disks
Capacity and Limitations	All files owned by root Read-only No compression
How to Use	`$ genromfs -f ./romfs-rfs/rfs -d $RFS` Create filesystem with mkromfs utility. Creating device nodes particularly interesting – create a file starting with @ with device node information. Example: @console,5,1
Home Page More Info	<kernel>/Documentation/filesystems http://romfs.sourceforge.net/ http://lldn.timesys.com/docs/tiny_flash

A Word About MTD

- **MTD "Memory Technology Device" is used for flash devices.**
  - **These are not block devices**
    - **/dev/mtdblockX serves as a primitive translation layer, but you shouldn't go putting a block-based filesystem on this device.**
  - **Not character devices either**
- **What's the difference**
  - **Work by manipulating "erase blocks"**
  - **Erase blocks then contain some number file nodes**
  - **Can "wear out", must spread writes over the media to avoid**
- **MTD vs. Flash Drives/USB Sticks**
  - **These devices contain a Flash Translation Layer that performs wear leveling and presents a block device.**
- **Use JFFS2 with devices that don't have a flash translation layer.**

JFFS2

Description	Read/Write filesystem designed specifically for MTD/Flash based devices. Handles wear leveling and compresses data during creation and subsequent writes
When to Use	Flash-based storage hardware
Capacity and Limitations	2^{32} GB, 2^{32} files, Page size from 2^{12} to 2^{18} Complete POSIX meta data Mounts slowly (improved lately); at capacity, writes can be slow
How to Use	`$ mkfs.jffs2 -o ../<bsp_name>-flash.jffs2 -e 00040000` Full details at: http://lldn.timesys.com/docs/jffs2 rootfstype=jffs2 on the command line, mounting the /dev/mtdblock device as the root device.
Home Page More Info	http://sourceware.org/jffs2 http://sourceware.org/jffs2/jffs2-html/jffs2-html.html http://lldn.timesys.com/tag/jffs2

YAFFS2

Description	Yet Another Flash FileSystem. Works, in principle, much like JFFS2, but designed specifically for NAND flash devices, which are a bit different than MTD flash devices.
When to Use	NAND flash devices
Capacity and Limitations	2^{32} GB, 2^{32} files Complete POSIX metadata No compression
How to Use	Filesystems created using user space tool, much like JFFS2. The resulting file can then be written directly to a flash partition.
Home Page More Info	http://www.aleph1.co.uk/taxonomy/term/31 http://www.aleph1.co.uk/node/40 http://lldn.timesys.com/docs/tiny_flash

initramfs

- Integral part of 2.6 Linux kernel boot
  - A filesystem that sits on top of the kernel's inode cache
  - Looks for initramfs before using "traditional booting method"
  - Can use as "real" filesystem
- How to create
  - Part of the kernel build process
  - As a compressed cpio archive

```
$ cd <rfs-directory>
$ find . | cpio -o -H newc | gzip > ../initramfs_data.cpio.gz
```

  - Point to a directory
    - Make CONFIG_INITRAMFS_SOURCE a directory name
  - Use specification file
    - Make CONFIG_INITRAMFS_SOURCE a file name that specifies what files\devices to create with what ownership permissions
- More Information
  - http://www.timesys.com/timesource/initramfs.htm
  - http://lldn.timesys.com/tag/initramfs

initramfs and Booting

1. At boot time, the kernel extracts an archive (cpio format) into a ramfs filesystem, called rootfs.
   - When this archive isn't present, an empty rootfs is created.
   - Root filesystems mount over rootfs.
2. The kernel looks at the filesystem for an init, and runs it if it exists.
3. Otherwise, the kernel follows the "prior" boot algorithm.

In Summary

- **Block devices**
  - ext2 – Very stable, easy to work with, widely supported, keeps all permissions… but, not very space efficient
  - cramfs – Produces a small filesystem … tradeoff: read-only with minimal permissions
  - squashfs – More metadata and larger filesystems, great compression results in small filesystem, but … performance hit
  - romfs – Small kernel module, but … lacks compression
- **Flash**
  - JFFS2 – Stores all metadata, high capacity … performance lacking on mount times and writes (under certain circumstances)
  - YAFFS2 – Handles particularities of NAND flash … performance also lacking under certain circumstances
- **In Memory**
  - initramfs – Complete support for permissions and file ownership, however … stored in memory, so changes aren't persistent

How These Stack Up in the Real World

- **Created filesystem**
  - **Busybox 1.2, statically linked, ~600K**
    - **Basic filesystem: init, some file tools, http server**
  - **Minimal devices**
  - **Did not size the filesystem any larger than necessary**
- **Results**

305,376	initramfs
306,992	squashfs
339,968	cramfs
358,608	JFFS2
686,400	YAFFS2
577,537	romfs
701,440	ext2

These results are less surprising than one would think. The read-only filesystems don't have as much overhead, and are, therefore, smaller. I could not figure out why YAFFS2 was so much larger. (Sorry!)

What about NFS?

- Rarely used in production systems
  - Great way for testing your board
  - Relying on network in production is risky
  - Not very fast on Linux, slow when using Cygwin as a server
- While a filesystem from a technical perspective, it is a protocol
  - Makes some filesystem remotely accessible
  - Negotiates privileges, what clients can access in the resource
  - Can export any filesystem type for access over NFS (well, almost any)

Not Mentioned

- ISO9660
  - Since this is a read-only filesystem, it could be put on a flash partition.
- vfat
  - Small, yes, but not space efficient. Has the extra baggage of the “case preserving” nature of the MS-DOS filesystem.
- minix fs
  - Maybe I should have. Simple and fast. Very small driver footprint, but no compression.

Recommendations

- **Read-Only**
  - squashfs – Best compression, ability to control endianness and compression
- **Flash**
  - JFFS2 – The standard, compresses well, well-supported and rock solid, recent improvements in performance, too!
- **Development**
  - NFS – Small impact on kernel size, can configure as read-only so it looks like system

Thank you for attending!