Cramfs - cram a filesystem onto a small ROM

cramfs is designed to be simple and small, and to compress things well.

It uses the zlib routines to compress a file one page at a time, and allows random page access. The meta-data is not compressed, but is expressed in a very terse representation to make it use much less diskspace than traditional filesystems.

You can't write to a cramfs filesystem (making it compressible and compact also makes it _very_ hard to update on-the-fly), so you have to create the disk image with the "mkcramfs" utility.

Usage Notes

File sizes are limited to less than 16MB.

Maximum filesystem size is a little over 256MB. (The last file on the filesystem is allowed to extend past 256MB.)

Only the low 8 bits of gid are stored. The current version of mkcramfs simply truncates to 8 bits, which is a potential security issue.

Hard links are supported, but hard linked files will still have a link count of 1 in the cramfs image.

Cramfs directories have no `.` or `..` entries. Directories (like every other file on cramfs) always have a link count of 1. (There's no need to use -noleaf in `find`, btw.)

No timestamps are stored in a cramfs, so these default to the epoch (1970 GMT). Recently-accessed files may have updated timestamps, but the update lasts only as long as the inode is cached in memory, after which the timestamp reverts to 1970, i.e. moves backwards in time.

Currently, cramfs must be written and read with architectures of the same endianness, and can be read only by kernels with PAGE_SIZE == 4096. At least the latter of these is a bug, but it hasn't been decided what the best fix is. For the moment if you have larger pages you can just change the #define in mkcramfs.c, so long as you don't mind the filesystem becoming unreadable to future kernels.

Memory Mapped cramfs image

The CRAMFS_MTD Kconfig option adds support for loading data directly from a physical linear memory range (usually non volatile memory like Flash) instead of going through the block device layer. This saves some memory since no intermediate buffering is necessary to hold the data before decompressing.

And when data blocks are kept uncompressed and properly aligned, they will automatically be mapped directly into user space whenever possible providing eXecute-In-Place (XIP) from ROM of read-only segments. Data segments mapped read-write (hence they have to be copied to RAM) may still be compressed in the cramfs image in the same file along with non compressed read-only segments. Both MMU and no-MMU systems are supported. This is particularly handy for tiny embedded systems with very tight memory constraints.

The location of the cramfs image in memory is system dependent. You must know the proper physical address where the cramfs image is located and configure an MTD device for it. Also, that MTD device must be supported by a map driver that implements the "point" method. Examples of such MTD drivers are cfi_cmdset_0001 (Intel/Sharp CFI flash) or physmap (Flash device in physical memory map). MTD partitions based on such devices are fine too. Then that device should be specified with the "mtd:" prefix as the mount device argument. For example, to mount the MTD device named "fs_partition" on the /mnt directory:

```
$ mount -t cramfs mtd:fs_partition /mnt
```

To boot a kernel with this as root filesystem, suffice to specify something like "root=mtd:fs_partition" on the kernel command line.

Tools

A version of mkcramfs that can take advantage of the latest capabilities described above can be found here:

https://github.com/npitre/cramfs-tools

For /usr/share/magic

0	ulelong 0x28cd3d45	Linux cramfs offset 0
>4	ulelong x	size %d
>8	ulelong x	flags 0x%x
>12	ulelong x	future 0x%x
>16	string >0	signature "%.16s"
>32	ulelong x	fsid.crc 0x%x
>36	ulelong x	fsid.edition %d

>40	ulelong x	fsid.blocks %d
>44	ulelong x	fsid.files %d
>48	string >0	name "%.16s"
512	ulelong 0x28cd3d45	Linux cramfs offset 512
>516	ulelong x	size %d
>520	ulelong x	flags 0x%x
>524	ulelong x	future 0x%x
>528	string >0	signature "%.16s"
>544	ulelong x	fsid.crc 0x%x
>548	ulelong x	fsid.edition %d
>552	ulelong x	fsid.blocks %d
>556	ulelong x	fsid.files %d
>560	string >0	name "%.16s"

Hacker Notes

See fs/cramfs/README for filesystem layout and implementation notes.