Execute-in-place for file mappings

Motivation

File mappings are performed by mapping page cache pages to userspace. In addition, read&write type file operations also transfer data from/to the page cache.

For memory backed storage devices that use the block device interface, the page cache pages are in fact copies of the original storage. Various approaches exist to work around the need for an extra copy. The ramdisk driver for example does read the data into the page cache, keeps a reference, and discards the original data behind later on.

Execute-in-place solves this issue the other way around: instead of keeping data in the page cache, the need to have a page cache copy is eliminated completely. With execute-in-place, read&write type operations are performed directly from/to the memory backed storage device. For file mappings, the storage device itself is mapped directly into userspace.

This implementation was initially written for shared memory segments between different virtual machines on s390 hardware to allow multiple machines to share the same binaries and libraries.

Implementation

Execute-in-place is implemented in three steps: block device operation, address space operation, and file operations.

A block device operation named direct_access is used to retrieve a reference (pointer) to a block on-disk. The reference is supposed to be cpu-addressable, physical address and remain valid until the release operation is performed. A struct block_device reference is used to address the device, and a sector_t argument is used to identify the individual block. As an alternative, memory technology devices can be used for this.

The block device operation is optional, these block devices support it as of today:
- dcssblk: s390 dcss block device driver

An address space operation named get_xip_mem is used to retrieve references to a page frame number and a kernel address. To obtain these values a reference to an address_space is provided. This function assigns values to the kmem and pfn parameters. The third argument indicates whether the function should allocate
blocks if needed.

This address space operation is mutually exclusive with readpage&writepage that do page cache read/write operations.
The following filesystems support it as of today:
- ext2: the second extended filesystem, see Documentation/filesystems/ext2.txt

A set of file operations that do utilize get_xip_page can be found in mm/filemap_xip.c . The following file operation implementations are provided:
- aio_read/aio_write

- readv/writev
- sendfile

The generic file operations do_sync_read/do_sync_write can be used to implement classic synchronous IO calls.

Shortcomings

This implementation is limited to storage devices that are cpu addressable at all times (no highmem or such). It works well on rom/ram, but enhancements are needed to make it work with flash in read+write mode.
Putting the Linux kernel and/or its modules on a xip filesystem does not mean they are not copied.