Lecture 13

File systems in Linux

- What is the Linux Virtual File System ?
- What is a superblock ?
- What is an i-node ?
- How is a path name translated to an i-node as part of the open() system call ?

Virtual file system

- Linux supports a wide range of file systems by working with two levels of abstraction.
- At the higher level, the Virtual File System (VFS) implements a number of common services and generic file operations.
- It is organised as a collection of base classes, one for each area of file system activity.
- For each part, the specific file system has a structure containing function pointers defining the operations it provides.
- Pointers to these structures are stored in the generic data structures representing mounted file systems, open files, etc.
- The starting point for associating file system specific operations with generic operations comes when a file system registers itself, passing a structure with a pointer to the function that loads the superblock when that file system is mounted.

Superblocks

- When a fs is mounted, a fs specific function is called to load an internal representation of the fs metadata. Named after the original UNIX on-disk metadata, this is called the *superblock*.
- A member of the internal superblock structure points to a structure of type *struct super_operations*. This structure contains a number of function pointers that are needed to carry out operations on a mounted fs.
- Although the name suggests that these functions are primarily related to the superblock, most are actually functions needed to fetch and manipulate other metadata structures called *i-nodes*.

i-node structure

- Some of the more interesting members of the structure include:
  - *alloc_inode()* – allocate the memory for and initialize an in-memory i-node structure;
  - *read_inode()* – read an i-node from the fs (disk);
  - *write_inode()* – write a modified i-node back to the file system;
  - *write_super()* – similarly handle a modified superblock;
  - *sync_fs()* – ensure that the fs as stored on the device is up to date with respect to any cached data.
- One of its members points to a structure which contains function pointers for the operations needed for operating on i-nodes or directories described by the i-node.

- Some of the relevant operations include:
  - create() – create a new file in a directory;
  - lookup() – fetch a directory entry from a directory;
  - mkdir() – create a new subdirectory;
  - getattr() – return metadata from an i-node.
- Directories are implemented as lists of directory entries. The VFS maintains a cache of directory entries that provide a mapping from file names to i-nodes. They can be quickly searched to avoid unnecessary disk accesses.
- In addition to the i-node operations, the internal i-node structure also points to a structure of type struct file_operations. When a file is opened, an internal structure representing the open file is created. This structure points to the file operations structure (open, read, write, ioctl,..).

The EXT3 file system

- The third extended (EXT3) fs is probably the most used file system in Linux.
- In any disk or partition holding the EXT3 fs, the first block is reserved for boot. The next block is a superblock, which is replicated in several places in the fs (the block size can be 1024, 2048, 4096 and even 8192 B).
- EXT3 divides the fs into block groups, each of them having a copy of the superblock. The following block is one-block group descriptor table, and then two blocks of free bitmaps – one for free blocks within the group and one for free i-nodes within the group. Following are i-node blocks. The rest of the blocks are for data.
- The strategy is to keep the blocks allocated to a file in the same group together with its i-node.
- Each entry of the directory structure contains the file name and the i-number. EXT3 provides an option to speedup directory searches by adding a hash table to the directory.
- EXT3 has a journal, stored as a regular file.

EXT3 name lookup

- The open() system call enters the kernel with the function sys_open(). The major function is to locate the i-node that corresponds to the path name passed by the application.
- The open sys call checks the permission of the operation with the i-node and then builds the internal open file data structure.
- After about seven nested function calls, the VFS function __link_path_walk() is invoked, to follow the path name along the directory tree.
- The path name can be either absolute or relative. For an absolute path name, one leading slash is sufficient. The first name points to the directory entry (which in turn points to the i-node) – this is the root directory, or the current directory.

Write operation

- When the file is open the application can call read() and write().
- Writing to a file begins by determining the point where writing starts (part of the open file structure). After that, the control is passed to VFS, where the request is checked to make sure that it doesn't violate security (e.g., file open read-only) or other limitations. Then, the control is passed to the EXT3 specific write function.

sys_file() → vfs_write() → ext3_file_write()