

CSIS0230A Principle of Operating Systems(Class A)
Notes for Tutorial 10
The ext2 Filesystems

For some time, Linux distributions use the ext2 filesystem to store all its files by default. Over the years, different features have been designed for ext2, and there is no sign ext2 will become extinct in the near future. This is especially true after the recent addition of journaling capability to the filesystem (not discussed here).

1. Creating a filesystem

To create an ext2 filesystem, one uses `/sbin/mke2fs`. It should be invoked as `mke2fs [options] device-file`. Here *device-file* should be a block device to hold the filesystem, e.g., `/dev/hda3`. You can give it a normal filename if you like, although `mke2fs` will give you a warning. There are several important options that you should know:

- `-c`: Check for bad blocks before creating the filesystem.
- `-b`: Specify the block size. Normally the block size is 1024. Other possible values are 2048 and 4096. If you expect files to be larger, then this number should also be larger.
- `-i <bytes-per-inode>`: When the ext2 filesystem is created, some blocks are allocated for use as inodes. Once the filesystem is created, the number cannot be changed. By default, one inode is created per 4096 bytes of *device-file*. So if you create a 1G filesystem, $1G/4096 = 262144$ inodes are allocated, and thus the filesystem can contain at most that number of files, symbolic links, devices, fifos, sockets and directories.

Other options are described in the manpage of `mke2fs`.

2. Mounting and unmounting a filesystem

To use a filesystem, it must be mounted. The root filesystem is mounted when the kernel is booted, which is specified in the LILO bootloader (i.e., in `lilo.conf`). Other filesystems can be mounted into any mounted directories, using the `mount` command:

```
mount <device-file> <mount-point>
```

For example, if you have created an ext2 filesystem in `/dev/hda5` and want to use it as your `/usr`, you will type `mount /dev/hda5 /usr`. To reverse the change, you should use

```
umount <mount-point>
```

However, this works only if there is currently no programs using any file within the filesystem. You can list all processes using a particular filesystem by using the `fuser` utility, like

```
fuser -m <mount-point>
```

This lists the process-id's of all the processes using the mount point, so that you can select to kill them to clear the way for `umount`.

3. Overall layout

Please refer to the lecture notes p. 10.15 to see the overall layout of an ext2 filesystem.

4. The superblock

The superblock is the very first thing within the filesystem. It is located at block 1, second only to the boot block. It contains control information about the whole filesystem. Here is the layout of the superblock:

	0x0-0x1	0x2-0x3	0x4-0x5	0x6-0x7	0x8-0x9	0xA-0xB	0xC-0xD	0xE-0xF
0x00	Inode count		Block count		Reserved block		Free block count	
0x01	Free inode count		First data block		log block size		log frag size*	
0x02	Block per group		*Frag per group		Inode per group		Mount time	
0x03	Write time		Mount count	Max mount	Signature	FS state	Error behavior	
0x03	Last check time		Max time to check		Other info, padding			

Some field needs special attention: The log block size is defined as the base-2 logarithm of the block size, in 1024-bytes unit. So the permissible values are 0, 1 and 2, with the default being 0 (i.e., 1024 bytes). All the time fields are the standard Unix representation of time, i.e., in the number of seconds after UTC Jan 1 12:00am has passed.

In the old (kernel 2.0.x or before) ext2 filesystem, superblocks can be found in the beginning of all block groups, as backup copies. These backup copies are updated whenever an filesystem check is done (by `e2fsck`). In newer versions, only some block groups have superblock (and group descriptors, see below) attached. This reduces filesystem check time, and also the storage used for the redundant information. When the filesystem is created, those blocks containing a backup superblock and group descriptors are printed to the screen. The second block group always has a backup copy, usually at block 8193 (if the block size is 1k). If your filesystem breaks seriously due to an improper shutdown, and most information about the disk are wrong, `fsck` will suggest you to use that backup superblock and group descriptors.

5. Block groups and group descriptors

The disk is divided into multiple block groups, so that each block group has a free block bitmap that can be held within one block. For example, if the block size is 1024 byte, there are $8 \times 1024 = 8192$ bits per block, each block group is thus of size 8192 blocks, each represented by a bit in the free block bitmap. The bit is 1 if the block is free, and 0 if otherwise.

Information about a block group is held in a group descriptor. It has the following format:

	0x0-0x1	0x2-0x3	0x4-0x5	0x6-0x7	0x8-0x9	0xA-0xB	0xC-0xD	0xE-0xF
0x00	Block bitmap block		Inode bitmap block		Inode table block		# Free blocks	# Free inodes
0x01	# Directory	Reserved						

So the group descriptor gives an easy way for the kernel to know the current number of blocks and inodes available and used, for the purpose of balancing the loading of the groups (i.e., how many files and directories are within each group). It also keeps the block numbers at which the bitmaps and tables are stored. As a rule, block number means the number of blocks from the beginning of the partition, not the group.

All block group descriptors are packed into one (or a few) blocks at the beginning of the first group, right after the superblock (i.e., at block 2). Like the superblock, backup copies can be found in some of the later groups.

6. Inode map and inode table

Each file has an inode associated, to hold all information about the file (except its filename, which is held as directory data). An inode is a 128-byte structure, so each 1024-byte block can hold 8 of them, and larger blocks can hold more. The information stored includes:

- The file type (i.e., plain file, directory, symbolic link, etc.)
- Permission information (uid, gid, file mode)
- File times (access, creation, modification and deletion time)
- File size
- Number of hard-links to the inode
- The block numbers of its first 12 blocks of file data (direct blocks).
- The block number of its indirect, double indirect and triple indirect blocks

Inodes spread evenly across all block groups. For example, if you have a 1G disk of block size 1k, each block group will be of size 8192 blocks, so there will be $1G/8192k = 128$ block groups. As we have seen, if standard ratio of 4k per inode is used, there will be 262144 inodes, and each block group will hold 2048 of them, in $2048/8 = 256$ blocks. They are held before the data blocks of each group.

Like data blocks, it is necessary to know whether an inode is free or not. This is again done using bitmaps. Each group has one block spared for inode bitmap, although typically not all bitmaps are filled. In the above example, we only have 262144 inodes, so only $262144/8192 = 32$ blocks are needed. So inode bitmaps of the remaining $256 - 32 = 224$ block groups are actually empty.

7. Directory entries

An ext2 directory entry is very simple. It is of variable size, but the size is always a multiple of 4 to make alignment easy.

	0x0-0x1	0x2-0x3	0x4-0x5	0x6-0x7	0x8-0x9	0xA-0xB	0xC-0xD	0xE-0xF
0x00	Inode number	Entry length	Name length	Filename (variable length)				