

Travel in VFS and Ext4

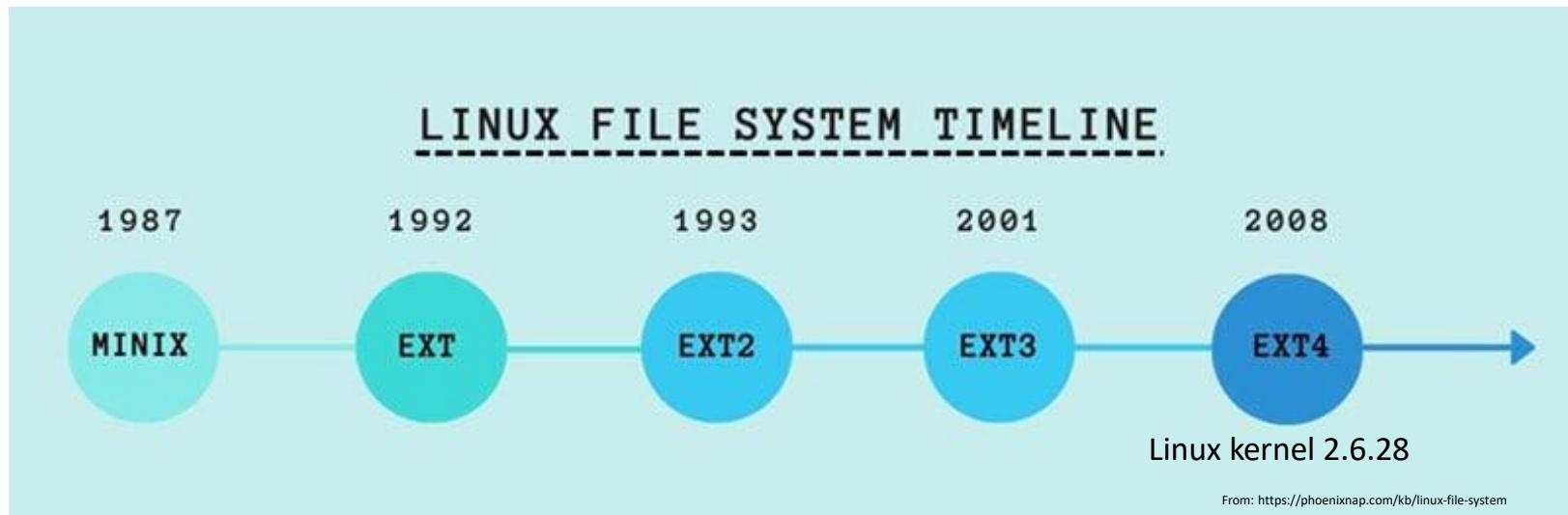
# Title and content

- What is Ext4
- VFS
- Superblock
- Disk Layout
- Inode
- Directory Entry
- File Object
- Debugfs – Prowl around a FS
- Syscalls

# What is Ext4

The Extended Filesystem 4 (ext4) or fourth extended filesystem is a [journaling file system](#) for [Linux](#), developed as the successor to [ext3](#).

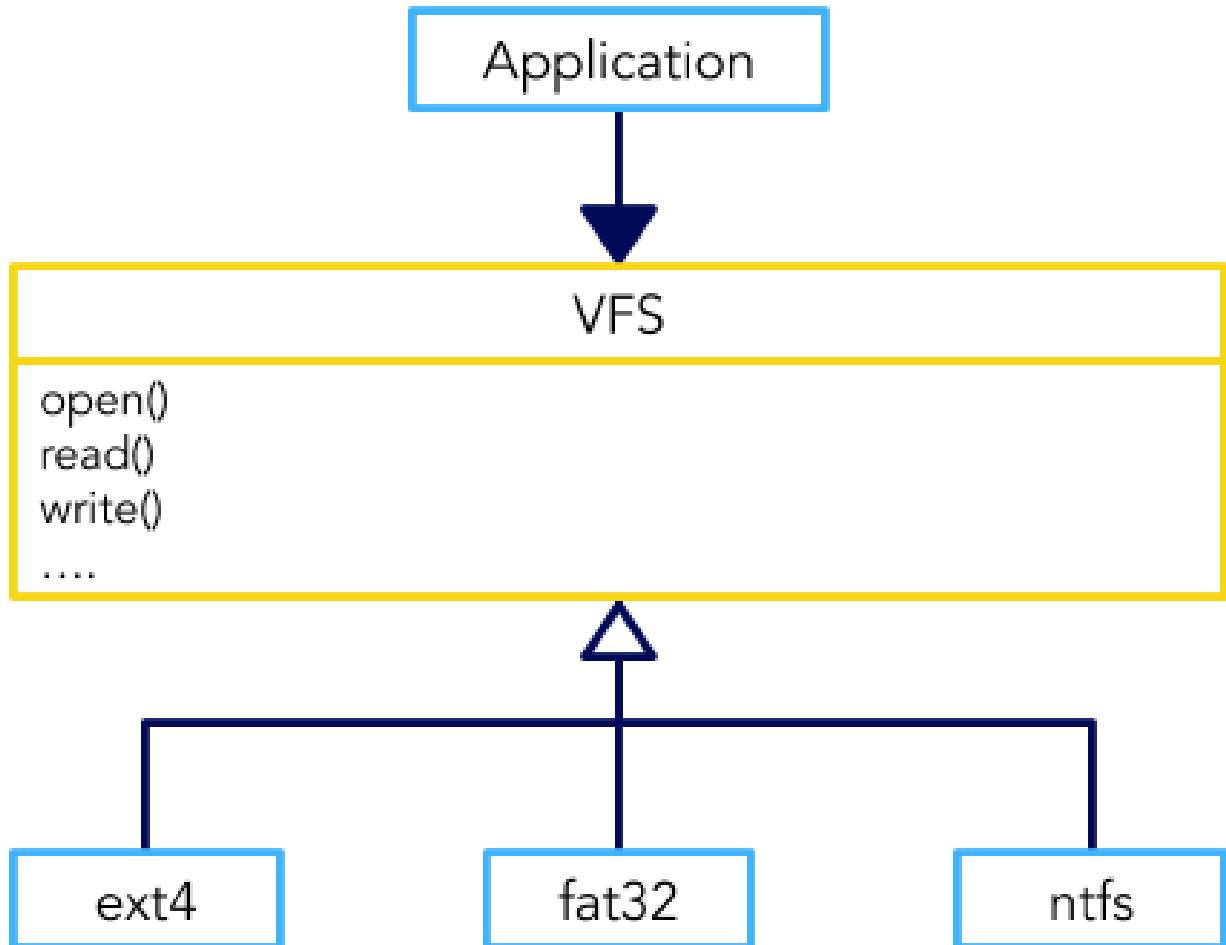
From <https://en.wikipedia.org/wiki/Ext4>



# Virtual File System - VFS

The Virtual File System (also known as the Virtual Filesystem Switch) is the software layer in the kernel that provides the filesystem interface to userspace programs.

It also provides an abstraction within the kernel which allows different filesystem implementations to coexist.



# Virtual File System - VFS

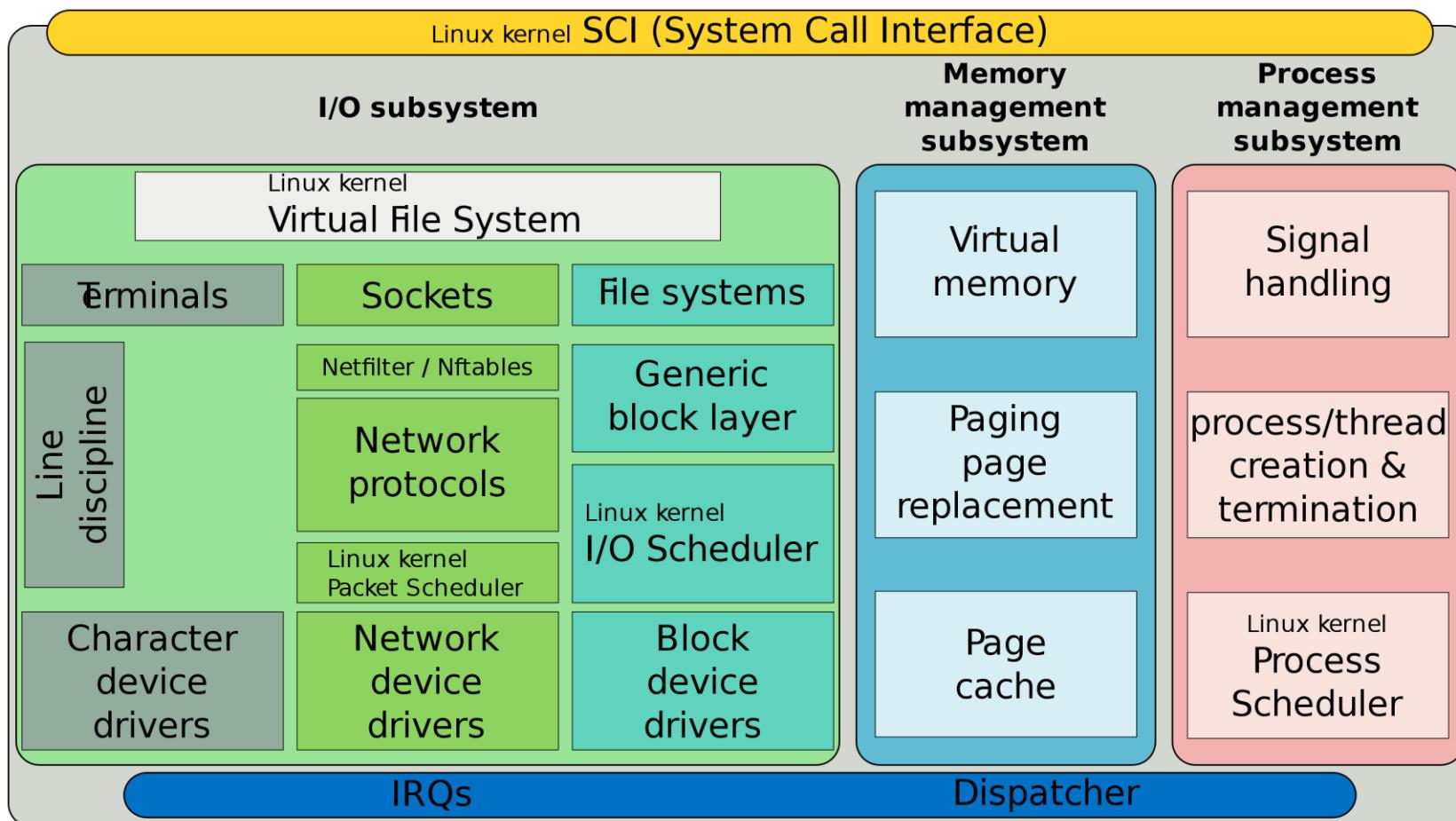
## 4 Main Abstracted Objects in VFS

- Superblock Object: representing a mounted filesystem.
- Inode Object: representing a file (device/FIFO/socket) in filesystem.
- Directory Entry Object: representing a directory entry (dentry).
- File Object: representing an opened file in process.

## Operations/Structs of VFS Object

- *super\_operations*: how the VFS can manipulate the superblock of your filesystem
- *inode\_operations*: how the VFS can manipulate an inode in your filesystem
- *dentry\_operations*: how a filesystem can overload the standard dentry operations
- *file\_operations*: how the VFS can manipulate an open file

# VFS & Ext4



From <https://en.wikipedia.org/wiki/Ext4>

Simplified structure of the Linux kernel: ext4 is implemented between the Linux kernel Virtual File System and the generic block layer.

# VFS

## Chapter 8: The Virtual Filesystem

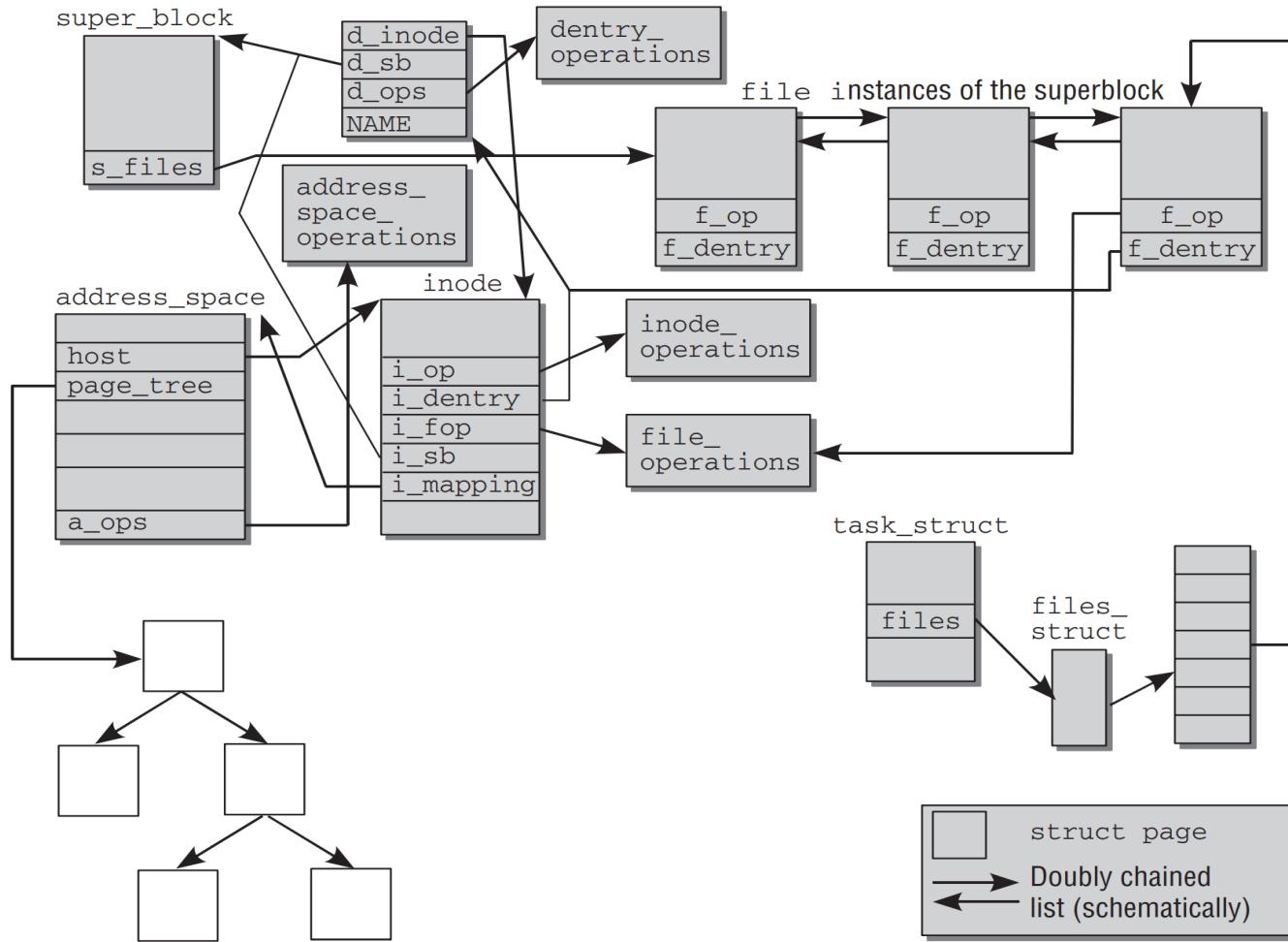


Figure 8-3: Interplay of the VFS components.

# Superblock

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ dumpe2fs -h mydisk
dumpe2fs 1.45.5 (07-Jan-2020)
Filesystem volume name: <none>
Last mounted on: /home/zegang/1300-simple-fs/mydisk_mp
Filesystem UUID: 8115e317-4965-4fec-9291-10d2497c8691
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg sparse_super large_file huge_file dir_nlink extra_isize metadata_csum
Filesystem flags: signed_directory_hash
Default mount options: user_xattr acl
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 32768
Block count: 32768
Reserved block count: 1638
Free blocks: 27620
Free inodes: 32754
First block: 0
Block size: 4096
Fragment size: 4096
Group descriptor size: 64
Reserved GDT blocks: 15
Blocks per group: 32768
Fragments per group: 32768
Inodes per group: 32768
Inode blocks per group: 1024
Flex block group size: 16
Filesystem created: Thu Feb 24 17:36:56 2022
Last mount time: Fri Feb 25 14:44:24 2022
Last write time: Fri Feb 25 14:44:24 2022
Mount count: 2
Maximum mount count: -1
Last checked: Thu Feb 24 17:36:56 2022
Check interval: 0 (<none>)
Lifetime writes: 289 kB
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 128
Journal inode: 8
Default directory hash: half_md4
Directory Hash Seed: 0ec78436-d75b-4d60-a4d4-e0b05cdcbfbb
Journal backup: inode blocks
Checksum type: crc32c
Checksum: 0xaf25dd46
Journal features: journal_64bit journal_checksum_v3
Journal size: 16M
Journal length: 4096
Journal sequence: 0x0000000b
Journal start: 1
Journal checksum type: crc32c
Journal checksum: 0x4a5386bd
```

# Superblock

A superblock is a record of the characteristics of a filesystem, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups.

```
$ dumpe2fs mydisk
```

```
Group 0: (Blocks 0-32767) csum 0x1894 [ITABLE_ZEROED]
Primary superblock at 0, Group descriptors at 1-1
Reserved GDT blocks at 2-16
Block bitmap at 17 (+17), csum 0x48de92ce
Inode bitmap at 33 (+33), csum 0x08c1e875
Inode table at 49-1072 (+49)
27108 free blocks, 32753 free inodes, 4 directories, 32753 unused inodes
Free blocks: 5660-32767
Free inodes: 16-32768
```

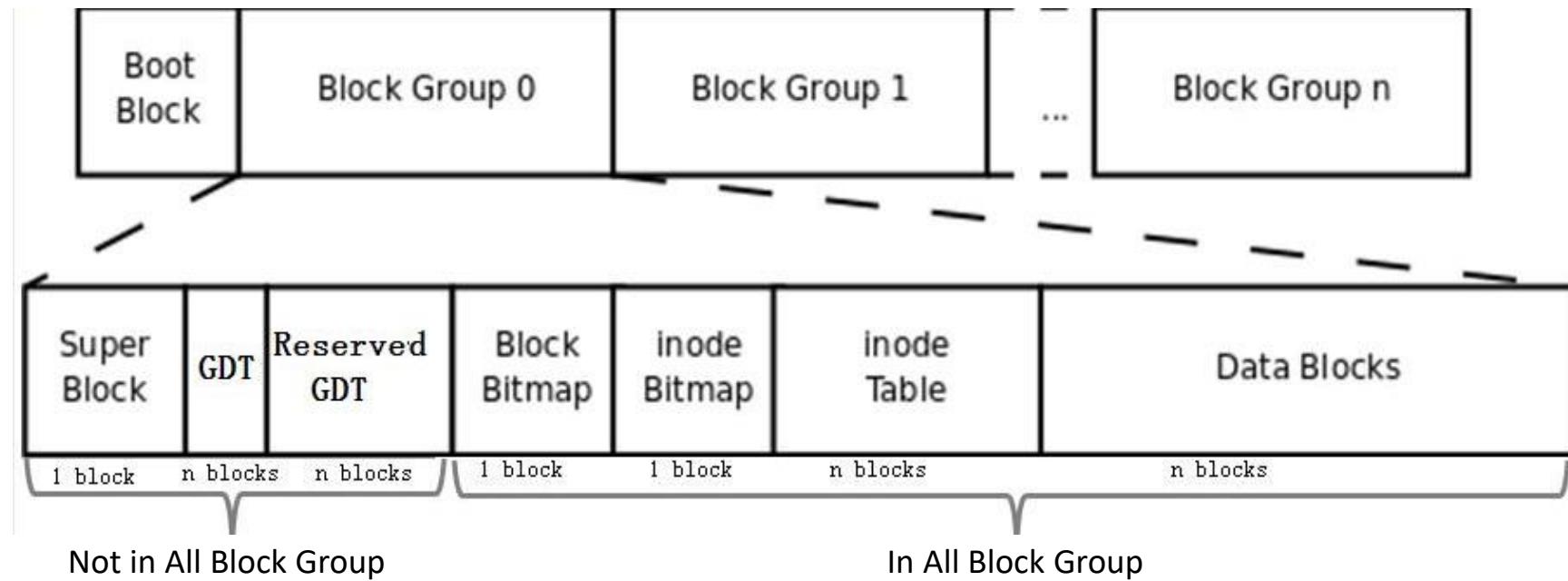
The superblock acquired its name from the fact that the first data block of a disk or of a partition was used to hold the metadata (i.e., data about data) about the partition itself.

# Superblock: Ext4 to VFS

```
1419 struct super_block {  
1420     struct list_head s_list;          /* Keep this first */  
1421     dev_t             s_dev;           /* search index; _not_ kdev_t */  
1422     unsigned char     s_blocksize_bits;  
1423     unsigned long     s_blocksize;  
1424     loff_t            s_maxbytes;      /* Max file size */  
1425     struct file_system_type *s_type;  
1426     const struct super_operations  *s_op;  
1427     const struct dquot_operations *dq_op;  
1428     const struct quotactl_ops    *s_qcop;  
1429     const struct export_operations *s_export_op;  
1430     unsigned long        s_flags;  
1431     unsigned long        s_iflags;       /* internal SB_I_* flags */  
1432     unsigned long        s_magic;  
1433     struct dentry         *s_root;  
1434     struct rw_semaphore   s_umount;  
1435     int                 s_count;  
1436     atomic_t            s_active;  
1437 #ifdef CONFIG_SECURITY  
1438     void                *s_security;  
1439 #endif  
1440     const struct xattr_handler **s_xattr;  
1441 #ifdef CONFIG_FS_ENCRYPTION  
1442     const struct fscrypt_operations *s_cop;  
1443     struct key            *s_master_keys; /* master crypto keys in use */  
1444 #endif  
1445 #ifdef CONFIG_FS_VERITY  
1446     const struct fsverity_operations *s_vop;  
1447 #endif  
1448  
1449  
https://elixir.bootlin.com/linux/v5.13/source/include/linux/fs.h#L1419
```

```
6636 static struct dentry *ext4_mount(struct file_system_type *fs_type, int flags,  
6637                                     const char *dev_name, void *data)  
6638 {  
6639     return mount_bdev(fs_type, flags, dev_name, data, ext4_fill_super);  
6640 }  
6641  
6642  
6643  
6644  
6645  
6646  
6647  
6648  
6649  
6650  
6651  
6652  
6653  
6654  
6655  
6656  
6657  
6658  
6659  
6660  
6661  
6662  
6663  
6664  
6665  
6666  
6667  
6668  
6669  
6670  
6671  
6672  
6673  
6674  
6675  
6676  
6677  
6678  
6679  
6680  
6681  
6682  
6683  
6684  
6685  
6686  
6687  
6688  
6689  
6690  
6691  
6692  
6693  
6694  
6695  
6696  
6697  
6698  
6699  
6700  
6701  
6702  
6703  
6704  
6705  
6706  
6707  
6708  
6709  
6710  
6711  
6712  
6713  
6714  
6715  
6716  
6717  
6718  
6719  
6720  
6721  
6722  
6723  
6724  
6725  
6726  
6727  
6728  
6729  
6730  
6731  
6732  
6733  
6734  
6735  
6736  
6737  
6738  
6739  
6740  
6741  
6742  
6743  
6744  
6745  
6746  
6747  
6748  
6749  
6750  
6751  
6752  
6753  
6754  
6755  
6756  
6757  
6758  
6759  
6760  
6761  
6762  
6763  
6764  
6765  
6766  
6767  
6768  
6769  
6770  
6771  
6772  
6773  
6774  
6775  
6776  
6777  
6778  
6779  
6780  
6781  
6782  
6783  
6784  
6785  
6786  
6787  
6788  
6789  
6790  
6791  
6792  
6793  
6794  
6795  
6796  
6797  
6798  
6799  
6800  
6801  
6802  
6803  
6804  
6805  
6806  
6807  
6808  
6809  
6810  
6811  
6812  
6813  
6814  
6815  
6816  
6817  
6818  
6819  
6820  
6821  
6822  
6823  
6824  
6825  
6826  
6827  
6828  
6829  
6830  
6831  
6832  
6833  
6834  
6835  
6836  
6837  
6838  
6839  
6840  
6841  
6842  
6843  
6844  
6845  
6846  
6847  
6848  
6849  
6850  
6851  
6852  
6853  
6854  
6855  
6856  
6857  
6858  
6859  
6860  
6861  
6862  
6863  
6864  
6865  
6866  
6867  
6868  
6869  
6870  
6871  
6872  
6873  
6874  
6875  
6876  
6877  
6878  
6879  
6880  
6881  
6882  
6883  
6884  
6885  
6886  
6887  
6888  
6889  
6890  
6891  
6892  
6893  
6894  
6895  
6896  
6897  
6898  
6899  
6900  
6901  
6902  
6903  
6904  
6905  
6906  
6907  
6908  
6909  
6910  
6911  
6912  
6913  
6914  
6915  
6916  
6917  
6918  
6919  
6920  
6921  
6922  
6923  
6924  
6925  
6926  
6927  
6928  
6929  
6930  
6931  
6932  
6933  
6934  
6935  
6936  
6937  
6938  
6939  
6940  
6941  
6942  
6943  
6944  
6945  
6946  
6947  
6948  
6949  
6950  
6951  
6952  
6953  
6954  
6955  
6956  
6957  
6958  
6959  
6960  
6961  
6962  
6963  
6964  
6965  
6966  
6967  
6968  
6969  
6970  
6971  
6972  
6973  
6974  
6975  
6976  
6977  
6978  
6979  
6980  
6981  
6982  
6983  
6984  
6985  
6986  
6987  
6988  
6989  
6990  
6991  
6992  
6993  
6994  
6995  
6996  
6997  
6998  
6999  
6999  
7000  
7001  
7002  
7003  
7004  
7005  
7006  
7007  
7008  
7009  
7009  
7010  
7011  
7012  
7013  
7014  
7015  
7016  
7017  
7018  
7019  
7019  
7020  
7021  
7022  
7023  
7024  
7025  
7026  
7027  
7028  
7029  
7029  
7030  
7031  
7032  
7033  
7034  
7035  
7036  
7037  
7038  
7039  
7039  
7040  
7041  
7042  
7043  
7044  
7045  
7046  
7047  
7048  
7049  
7049  
7050  
7051  
7052  
7053  
7054  
7055  
7056  
7057  
7058  
7059  
7059  
7060  
7061  
7062  
7063  
7064  
7065  
7066  
7067  
7068  
7069  
7069  
7070  
7071  
7072  
7073  
7074  
7075  
7076  
7077  
7078  
7079  
7079  
7080  
7081  
7082  
7083  
7084  
7085  
7086  
7087  
7088  
7089  
7089  
7090  
7091  
7092  
7093  
7094  
7095  
7096  
7097  
7098  
7099  
7099  
7100  
7101  
7102  
7103  
7104  
7105  
7106  
7107  
7108  
7109  
7109  
7110  
7111  
7112  
7113  
7114  
7115  
7116  
7117  
7118  
7119  
7119  
7120  
7121  
7122  
7123  
7124  
7125  
7126  
7127  
7128  
7129  
7129  
7130  
7131  
7132  
7133  
7134  
7135  
7136  
7137  
7138  
7139  
7139  
7140  
7141  
7142  
7143  
7144  
7145  
7146  
7147  
7148  
7149  
7149  
7150  
7151  
7152  
7153  
7154  
7155  
7156  
7157  
7158  
7159  
7159  
7160  
7161  
7162  
7163  
7164  
7165  
7166  
7167  
7168  
7169  
7169  
7170  
7171  
7172  
7173  
7174  
7175  
7176  
7177  
7178  
7179  
7179  
7180  
7181  
7182  
7183  
7184  
7185  
7186  
7187  
7188  
7189  
7189  
7190  
7191  
7192  
7193  
7194  
7195  
7196  
7197  
7198  
7199  
7199  
7200  
7201  
7202  
7203  
7204  
7205  
7206  
7207  
7208  
7209  
7209  
7210  
7211  
7212  
7213  
7214  
7215  
7216  
7217  
7218  
7219  
7219  
7220  
7221  
7222  
7223  
7224  
7225  
7226  
7227  
7228  
7229  
7229  
7230  
7231  
7232  
7233  
7234  
7235  
7236  
7237  
7238  
7239  
7239  
7240  
7241  
7242  
7243  
7244  
7245  
7246  
7247  
7248  
7249  
7249  
7250  
7251  
7252  
7253  
7254  
7255  
7256  
7257  
7258  
7259  
7259  
7260  
7261  
7262  
7263  
7264  
7265  
7266  
7267  
7268  
7269  
7269  
7270  
7271  
7272  
7273  
7274  
7275  
7276  
7277  
7278  
7279  
7279  
7280  
7281  
7282  
7283  
7284  
7285  
7286  
7287  
7288  
7289  
7289  
7290  
7291  
7292  
7293  
7294  
7295  
7296  
7297  
7298  
7299  
7299  
7300  
7301  
7302  
7303  
7304  
7305  
7306  
7307  
7308  
7309  
7309  
7310  
7311  
7312  
7313  
7314  
7315  
7316  
7317  
7318  
7319  
7319  
7320  
7321  
7322  
7323  
7324  
7325  
7326  
7327  
7328  
7329  
7329  
7330  
7331  
7332  
7333  
7334  
7335  
7336  
7337  
7338  
7339  
7339  
7340  
7341  
7342  
7343  
7344  
7345  
7346  
7347  
7348  
7349  
7349  
7350  
7351  
7352  
7353  
7354  
7355  
7356  
7357  
7358  
7359  
7359  
7360  
7361  
7362  
7363  
7364  
7365  
7366  
7367  
7368  
7369  
7369  
7370  
7371  
7372  
7373  
7374  
7375  
7376  
7377  
7378  
7379  
7379  
7380  
7381  
7382  
7383  
7384  
7385  
7386  
7387  
7388  
7389  
7389  
7390  
7391  
7392  
7393  
7394  
7395  
7396  
7397  
7398  
7399  
7399  
7400  
7401  
7402  
7403  
7404  
7405  
7406  
7407  
7408  
7409  
7409  
7410  
7411  
7412  
7413  
7414  
7415  
7416  
7417  
7418  
7419  
7419  
7420  
7421  
7422  
7423  
7424  
7425  
7426  
7427  
7428  
7429  
7429  
7430  
7431  
7432  
7433  
7434  
7435  
7436  
7437  
7438  
7439  
7439  
7440  
7441  
7442  
7443  
7444  
7445  
7446  
7447  
7448  
7449  
7449  
7450  
7451  
7452  
7453  
7454  
7455  
7456  
7457  
7458  
7459  
7459  
7460  
7461  
7462  
7463  
7464  
7465  
7466  
7467  
7468  
7469  
7469  
7470  
7471  
7472  
7473  
7474  
7475  
7476  
7477  
7478  
7479  
7479  
7480  
7481  
7482  
7483  
7484  
7485  
7486  
7487  
7488  
7489  
7489  
7490  
7491  
7492  
7493  
7494  
7495  
7496  
7497  
7498  
7499  
7499  
7500  
7501  
7502  
7503  
7504  
7505  
7506  
7507  
7508  
7509  
7509  
7510  
7511  
7512  
7513  
7514  
7515  
7516  
7517  
7518  
7519  
7519  
7520  
7521  
7522  
7523  
7524  
7525  
7526  
7527  
7528  
7529  
7529  
7530  
7531  
7532  
7533  
7534  
7535  
7536  
7537  
7538  
7539  
7539  
7540  
7541  
7542  
7543  
7544  
7545  
7546  
7547  
7548  
7549  
7549  
7550  
7551  
7552  
7553  
7554  
7555  
7556  
7557  
7558  
7559  
7559  
7560  
7561  
7562  
7563  
7564  
7565  
7566  
7567  
7568  
7569  
7569  
7570  
7571  
7572  
7573  
7574  
7575  
7576  
7577  
7578  
7579  
7579  
7580  
7581  
7582  
7583  
7584  
7585  
7586  
7587  
7588  
7589  
7589  
7590  
7591  
7592  
7593  
7594  
7595  
7596  
7597  
7598  
7599  
7599  
7600  
7601  
7602  
7603  
7604  
7605  
7606  
7607  
7608  
7609  
7609  
7610  
7611  
7612  
7613  
7614  
7615  
7616  
7617  
7618  
7619  
7619  
7620  
7621  
7622  
7623  
7624  
7625  
7626  
7627  
7628  
7629  
7629  
7630  
7631  
7632  
7633  
7634  
7635  
7636  
7637  
7638  
7639  
7639  
7640  
7641  
7642  
7643  
7644  
7645  
7646  
7647  
7648  
7649  
7649  
7650  
7651  
7652  
7653  
7654  
7655  
7656  
7657  
7658  
7659  
7659  
7660  
7661  
7662  
7663  
7664  
7665  
7666  
7667  
7668  
7669  
7669  
7670  
7671  
7672  
7673  
7674  
7675  
7676  
7677  
7678  
7679  
7679  
7680  
7681  
7682  
7683  
7684  
7685  
7686  
7687  
7688  
7689  
7689  
7690  
7691  
7692  
7693  
7694  
7695  
7696  
7697  
7698  
7699  
7699  
7700  
7701  
7702  
7703  
7704  
7705  
7706  
7707  
7708  
7709  
7709  
7710  
7711  
7712  
7713  
7714  
7715  
7716  
7717  
7718  
7719  
7719  
7720  
7721  
7722  
7723  
7724  
7725  
7726  
7727  
7728  
7729  
7729  
7730  
7731  
7732  
7733  
7734  
7735  
7736  
7737  
7738  
7739  
7739  
7740  
7741  
7742  
7743  
7744  
7745  
7746  
7747  
7748  
7749  
7749  
7750  
7751  
7752  
7753  
7754  
7755  
7756  
7757  
7758  
7759  
7759  
7760  
7761  
7762  
7763  
7764  
7765  
7766  
7767  
7768  
7769  
7769  
7770  
7771  
7772  
7773  
7774  
7775  
7776  
7777  
7778  
7779  
7779  
7780  
7781  
7782  
7783  
7784  
7785  
7786  
7787  
7788  
7789  
7789  
7790  
7791  
7792  
7793  
7794  
7795  
7796  
7797  
7798  
7799  
7799  
7800  
7801  
7802  
7803  
7804  
7805  
7806  
7807  
7808  
7809  
7809  
7810  
7811  
7812  
7813  
7814  
7815  
7816  
7817  
7818  
7819  
7819  
7820  
7821  
7822  
7823  
7824  
7825  
7826  
7827  
7828  
7829  
7829  
7830  
7831  
7832  
7833  
7834  
7835  
7836  
7837  
7838  
7839  
7839  
7840  
7841  
7842  
7843  
7844  
7845  
7846  
7847  
7848  
7849  
7849  
7850  
7851  
7852  
7853  
7854  
7855  
7856  
7857  
7858  
7859  
7859  
7860  
7861  
7862  
7863  
7864  
7865  
7866  
7867  
7868  
7869  
7869  
7870  
7871  
7872  
7873  
7874  
7875  
7876  
7877  
7878  
7879  
7879  
7880  
7881  
7882  
7883  
7884  
7885  
7886  
7887  
7888  
7889  
7889  
7890  
7891  
7892  
7893  
7894  
7895  
7896  
7897  
7898  
7899  
7899  
7900  
7901  
7902  
7903  
7904  
7905  
7906  
7907  
7908  
7909  
7909  
7910  
7911  
7912  
7913  
7914  
7915  
7916  
7917  
7918  
7919  
7919  
7920  
7921  
7922  
7923  
7924  
7925  
7926  
7927  
7928  
7929  
7929  
7930  
7931  
7932  
7933  
7934  
7935  
7936  
7937  
7938  
7939  
7939  
7940  
7941  
7942  
7943  
7944  
7945  
7946  
7947  
7948  
7949  
7949  
7950  
7951  
7952  
7953  
7954  
7955  
7956  
7957  
7958  
7959  
7959  
7960  
7961  
7962  
7963  
7964  
7965  
7966  
7967  
7968  
7969  
7969  
7970  
7971  
7972  
7973  
7974  
7975  
7976  
7977  
7978  
7979  
7979  
7980  
7981  
7982  
7983  
7984  
7985  
7986  
7987  
7988  
7989  
7989  
7990  
7991  
7992  
7993  
7994  
7995  
7996  
7997  
7998  
7999  
7999  
8000  
8001  
8002  
8003  
8004  
8005  
8006  
8007  
8008  
8009  
8009  
8010  
8011  
8012  
8013  
8014  
8015  
8016  
8017  
8018  
8019  
8019  
8020  
8021  
8022  
8023  
8024  
8025  
8026  
8027  
8028  
8029  
8029  
8030  
8031  
8032  
8033  
8034  
8035  
8036  
8037  
8038  
8039  
8039  
8040  
8041  
8042  
8043  
8044  
8045  
8046  
8047  
8048  
8049  
8049  
8050  
8051  
8052  
8053  
8054  
8055  
8056  
8057  
8058  
8059  
8059  
8060  
8061  
8062  
8063  
8064  
8065  
8066  
8067  
8068  
8069  
8069  
8070  
8071  
8072  
8073  
8074  
8075  
8076  
8077  
8078  
8079  
8079  
8080  
8081  
8082  
8083  
8084  
8085  
8086  
8087  
8088  
8089  
8089  
8090  
8091  
8092  
8093  
8094  
8095  
8096  
8097  
8098  
8099  
8099  
8100  
8101  
8102  
8103  
8104  
8105  
8106  
8107  
8108  
8109  
8109  
8110  
8111  
8112  
8113  
8114  
8115  
8116  
8117  
8118  
8119  
8119  
8120  
8121  
8122  
8123  
8124  
8125  
8126  
8127  
8128  
8129  
8129  
8130  
8131  
8132  
8133  
8134  
8135  
8136  
8137  
8138  
8139  
8139  
8140  
8141  
8142  
8143  
8144  
8145  
8146  
8147  
8148  
8149  
8149  
8150  
8151  
8152  
8153  
8154  
8155  
8156  
8157  
8158  
8159  
8159  
8160  
8161  
8162  
8163  
8164  
8165  
8166  
8167  
8168  
8169  
8169  
8170  
8171  
8172  
8173  
8174  
8175  
8176  
8177  
8178  
8179  
8179  
8180  
8181  
8182  
8183  
8184  
8185  
8186  
8187  
8188  
8189  
8189  
8190  
8191  
8192  
8193  
8194  
8195  
8196  
8197  
8198  
8199  
8199  
8200  
8201  
8202  
8203  
8204  
8205  
8206  
8207  
8208  
8209  
8209  
8210  
8211  
8212  
8213  
8214  
8215  
8216  
8217  
8218  
8219  
8219  
8220  
8221  
8222  
8223  
8224  
8225  
8226  
8227  
8228  
8229  
8229  
8230  
8231  
8232  
8233  
8234  
8235  
8236  
8237  
8238  
8239  
8239  
8240  
8241  
8242  
8243  
8244  
8245  
8246  
8247  
8248  
8249  
8249  
8250  
8251  
8252  
8253  
825
```

# Disk Layout



The layout of a standard block group is approximately as follows:

Group 0 Padding	ext4 Super Block	Group Descriptors	Reserved GDT Blocks	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks
1024 bytes	1 block	many blocks	many blocks	1 block	1 block	many blocks	many more blocks

# Superblock: Ext4

```
1293 /*  
1294 * Structure of the super block  
1295 */  
1296 struct ext4_super_block {  
1297 /*00*/ __le32 s_inodes_count; /* Inodes count */  
1298 __le32 s_blocks_count_lo; /* Blocks count */  
1299 __le32 s_r_blocks_count_lo; /* Reserved blocks count */  
1300 __le32 s_free_blocks_count_lo; /* Free blocks count */  
1301 /*10*/ __le32 s_file_blocks_count; /* File blocks count */  
1302 __le32 s_first_data_block; /* First Data Block */  
1303 __le32 s_log_block_size; /* Block size */  
1304 __le32 s_log_cluster_size; /* Allocation cluster size */  
1305 /*20*/ __le32 s_blocks_per_group; /* # Blocks per group */  
1306 __le32 s_clusters_per_group; /* # Clusters per group */  
1307 __le32 s_inodes_per_group; /* # Inodes per group */  
1308 __le32 s_mtime; /* Mount time */  
1309 /*30*/ __le32 s_wtime; /* Write time */  
1310 __le16 s_mnt_count; /* Mount count */  
1311 __le16 s_max_mnt_count; /* Maximal mount count */  
1312 __le16 s_magic; /* Magic signature */  
1313 /*40*/ __le16 s_state; /* File system state */  
1314 __le16 s_errors; /* Behaviour when detecting errors */  
1315 __le16 s_minor_rev_level; /* minor revision Level */  
  
zegang@zegang-VirtualBox:~/1300-simple-fs$ dumpe2fs mydisk  
dumpe2fs 1.45.5 (07-Jan-2020)  
Filesystem volume name: <none>  
Last mounted on: /home/zegang/1300-simple-fs/mydisk_mp  
Filesystem UUID: 8115e317-4965-4fec-9291-10d2497c8691  
Filesystem magic number: 0xEF53  
Filesystem revision #: 1 (dynamic)  
Filesystem features: has_journal ext_attr resize_inode dir  
Filesystem flags: signed_directory_hash  
Default mount options: user_xattr acl  
Filesystem state: clean  
Errors behavior: Continue  
Filesystem OS type: Linux  
Inode count: 32768  
Block count: 32768  
Reserved block count: 1638  
Free blocks: 27108  
Free inodes: 32753  
First block: 0  
Block size: 4096  
Fragment size: 4096
```

```
/ include/uapi/linux/magic.h  
26 #define EXT4_SUPER_MAGIC 0xEF53  
27 #define BTRFS_SUPER_MAGIC 0x9123683E
```

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ hexdump -s 1024 -n 64 mydisk  
0000400 8000 0000 8000 0000 0666 0000 69e4 0000  
0000410 7ff1 0000 0000 0000 0002 0000 0002 0000  
0000420 8000 0000 8000 0000 8000 0000 6217 621c  
0000430 64ea 6223 0003 ffff ef53 0001 0001 0000  
0000440
```

# Superblock: Operations

/ include / linux / fs.h

```
2144  
2145  
2146 struct super_operations {  
2147     struct inode *(*alloc_inode)(struct super_block *sb);  
2148     void (*destroy_inode)(struct inode *);  
2149     void (*free_inode)(struct inode *);  
2150  
2151     void (*dirty_inode) (struct inode *, int flags);  
2152     int (*write_inode) (struct inode *, struct writeback_control *wbc);  
2153     int (*drop_inode) (struct inode *);  
2154     void (*evict_inode) (struct inode *);  
2155     void (*put_super) (struct super_block *);  
2156     int (*sync_fs)(struct super_block *sb, int wait);  
2157     int (*freeze_super) (struct super_block *);  
2158     int (*freeze_fs) (struct super_block *);  
2159     int (*thaw_super) (struct super_block *);  
2160     int (*unfreeze_fs) (struct super_block *);  
2161     int (*statfs) (struct dentry *, struct kstatfs *);  
2162     int (*remount_fs) (struct super_block *, int *, char *);  
2163     void (*umount_begin) (struct super_block *);  
2164  
2165     int (*show_options)(struct seq_file *, struct dentry *);  
2166     int (*show_devname)(struct seq_file *, struct dentry *);  
2167     int (*show_path)(struct seq_file *, struct dentry *);  
2168     int (*show_stats)(struct seq_file *, struct dentry *);  
2169 #ifdef CONFIG_QUOTA  
2170     ssize_t (*quota_read)(struct super_block *, int, char *, size_t, loff_t);  
2171     ssize_t (*quota_write)(struct super_block *, int, const char *, size_t, loff_t);  
2172     struct dqquot **(*get_dquots)(struct inode *);  
2173 #endif  
2174     int (*bdev_try_to_free_page)(struct super_block*, struct page*, gfp_t);  
2175     long (*nr_cached_objects)(struct super_block *,  
2176                               struct shrink_control *);  
2177     long (*free_cached_objects)(struct super_block *,  
2178                               struct shrink_control *);  
2179 };  
2180
```

/ fs / ext4 / super.c

```
1638 static const struct super_operations ext4_sops = {  
1639     .alloc_inode    = ext4_alloc_inode,  
1640     .free_inode     = ext4_free_in_core_inode,  
1641     .destroy_inode  = ext4_destroy_inode,  
1642     .write_inode    = ext4_write_inode,  
1643     .dirty_inode    = ext4_dirty_inode,  
1644     .drop_inode     = ext4_drop_inode,  
1645     .evict_inode    = ext4_evict_inode,  
1646     .put_super      = ext4_put_super,  
1647     .sync_fs        = ext4_sync_fs,  
1648     .freeze_fs      = ext4_freeze,  
1649     .unfreeze_fs   = ext4_unfreeze,  
1650     .statfs         = ext4_statfs,  
1651     .remount_fs    = ext4_remount,  
1652     .show_options   = ext4_show_options,  
1653 #ifdef CONFIG_QUOTA  
1654     .quota_read     = ext4_quota_read,  
1655     .quota_write    = ext4_quota_write,  
1656     .get_dquots    = ext4_get_dquots,  
1657 #endif  
1658     .bdev_try_to_free_page = bdev_try_to_free_page,  
1659 };
```

# Inode

An inode is a data structure on a filesystem on a Unix-like operating system that stores all the information about a file except its name and its actual data.

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ stat mydisk_mp/home/zegang/1.txt
  File: mydisk_mp/home/zegang/1.txt
  Size: 2          Blocks: 8          IO Block: 4096   regular file
Device: 70eh/1806d  Inode: 14          Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  zegang)  Gid: ( 1000/  zegang)
Access: 2022-02-24 17:56:53.000000000 +0800
Modify: 2022-02-24 17:56:44.000000000 +0800
Change: 2022-02-24 17:56:44.000000000 +0800
 Birth: -
zegang@zegang-VirtualBox:~/1300-simple-fs$ ls -il mydisk_mp/home/zegang/1.txt
14 -rw-rw-r-- 1 zegang zegang 2 月 24 17:56 mydisk_mp/home/zegang/1.txt
```

Metadata includes:

- the size of the file (in bytes) and its physical location,
- the file's owner and group,
- the file's access permissions,
- timestamps telling when the inode was created, last modified and last accessed,
- a reference count telling how many hard links point to the inode.

# Inode

Space for inodes must be set aside when an operating system (or a new filesystem) is installed, and that system does its initial structuring of the filesystem. Within any filesystem, the maximum number of inodes, and hence the maximum number of files, is set when the filesystem is created.

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ mkfs.ext4 mydisk
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 32768 4k blocks and 32768 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

## **-i bytes-per-inode**

Specify the bytes/inode ratio. `mke2fs` creates an inode for every `bytes-per-inode` bytes of space on the disk. The larger the `bytes-per-inode` ratio, the fewer inodes will be created. This value generally shouldn't be smaller than the blocksize of the filesystem, since in that case more inodes would be made than can ever be used. Be warned that it is not possible to change this ratio on a filesystem after it is created, so be careful deciding the correct value for this parameter. Note that resizing a filesystem changes the number of inodes to maintain this ratio.

## **-I inode-size**

Specify the size of each inode in bytes. The `inode-size` value must be a power of 2 larger or equal to 128. The larger the `inode-size` the more space the inode table will consume, and this reduces the usable space in the filesystem and can also negatively impact performance. It is not possible to change this value after the filesystem is created.

In kernels after 2.6.10 and some earlier vendor kernels it is possible to utilize inodes larger than 128 bytes to store extended attributes for improved performance. Extended attributes stored in large inodes are not visible with older kernels, and such filesystems will not be mountable with 2.4 kernels at all.

The default inode size is controlled by the `mke2fs.conf(5)` file. In the `mke2fs.conf` file shipped with `e2fsprogs`, the default inode size is 256 bytes for most file systems, except for small file systems where the inode size will be 128 bytes.

# Inode

```
605
606  /*
607   * Keep mostly read-only and often accessed (especially for
608   * the RCU path Lookup and 'stat' data) fields at the beginning
609   * of the 'struct inode'
610   */
611 struct inode {
612     umode_t          i_mode;
613     unsigned short   i_opflags;
614     kuid_t           i_uid;
615     kgid_t           i_gid;
616     unsigned int     i_flags;
617
618 #ifdef CONFIG_FS_POSIX_ACL
619     struct posix_acl *i_acl;
620     struct posix_acl *i_default_acl;
621 #endif
622
623     const struct inode_operations *i_op;
624     struct super_block   *i_sb;
625     struct address_space *i_mapping;
626
627 #ifdef CONFIG_SECURITY
628     void             *i_security;
629 #endif
630
631     /* Stat data, not accessed from path walking */
632     unsigned long     i_ino;
633     /*
634      * Filesystems may only read i_nlink directly. They shall use the
635      * following functions for modification:
636      *
637      *   (set/clear/inc/drop)_nlink
638      *   inode_(inc/dec)_Link_count
639      */
640     union {
641         const unsigned int i_nlink;
642         unsigned int __i_nlink;
643     };
644     dev_t             i_rdev;
645    loff_t            i_size;
646     struct timespec64 i_atime;
647     struct timespec64 i_mtime;
648     struct timespec64 i_ctime;
```

```
996
997 /*
998 * fourth extended file system inode data in memory
999 */
1000 struct ext4_inode_info {
1001     __le32 i_data[15];      /* unconverted */
1002     __u32 i_dtime;
1003     ext4_fsblk_t i_file_acl;
1004
1005     /*
1006     * i_block_group is the number of the block group which contains
1007     * the inode
1008     */
1009     struct rw_semaphore i_mmap_sem;
1010     struct inode vfs_inode;
1011     struct jbd2_inode *jinode;
```

```
/ include / trace / events / ext4.h

16 struct mp_page_da_data;
17 struct ext4_map_blocks;
18 struct extent_status;
19 struct ext4_fsmap;
20 struct partial_cluster;
21
22 #define EXT4_I(inode) (container_of(inode, struct ext4_inode_info, vfs_inode))
```

```
770  /*
771   * Structure of an inode on the disk
772   */
773  struct ext4_inode {
774      __le16  i_mode;          /* File mode */
775      __le16  i_uid;           /* Low 16 bits of Owner Uid */
776      __le32  i_size_lo;       /* Size in bytes */
777      __le32  i_atime;         /* Access time */
778      __le32  i_ctime;         /* Inode Change time */
```

# Inode - VFS APIs & Ext4

```
2063  
2064 struct inode_operations {  
2065     struct dentry * (*lookup) (struct inode *, struct dentry *, unsigned int);  
2066     const char * (*get_link) (struct dentry *, struct inode *, struct delayed_call *);  
2067     int (*permission) (struct user_namespace *, struct inode *, int);  
2068     struct posix_acl * (*get_acl)(struct inode *, int);  
2069  
2070     int (*readlink) (struct dentry *, char __user *, int);  
2071  
2072     int (*create) (struct user_namespace *, struct inode *, struct dentry *,  
2073                     umode_t, bool);  
2074     int (*link) (struct dentry *, struct inode *, struct dentry *);  
2075     int (*unlink) (struct inode *, struct dentry *);  
2076     int (*symlink) (struct user_namespace *, struct inode *, struct dentry *,  
2077                      const char *);  
2078     int (*mkdir) (struct user_namespace *, struct inode *, struct dentry *,  
2079                     umode_t);  
2080     int (*rmdir) (struct inode *, struct dentry *);  
2081     int (*mknod) (struct user_namespace *, struct inode *, struct dentry *,  
2082                     umode_t, dev_t);  
2083     int (*rename) (struct user_namespace *, struct inode *, struct dentry *,  
2084                     struct inode *, struct dentry *, unsigned int);  
2085     int (*setattr) (struct user_namespace *, struct dentry *,  
2086                     struct iattr *);  
2087     int (*getattr) (struct user_namespace *, const struct path *,  
2088                     struct kstat *, u32, unsigned int);  
2089     ssize_t (*listxattr) (struct dentry *, char *, size_t);  
2090     int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,  
2091                     u64 len);  
2092     int (*update_time)(struct inode *, struct timespec64 *, int);  
2093     int (*atomic_open)(struct inode *, struct dentry *,  
2094                         struct file *, unsigned open_flag,  
2095                         umode_t create_mode);  
2096     int (*tmpfile) (struct user_namespace *, struct inode *,  
2097                     struct dentry *, umode_t);  
2098     int (*set_acl)(struct user_namespace *, struct inode *,  
2099                     struct posix_acl *, int);  
2100     int (*fileattr_set)(struct user_namespace *mnt_userns,  
2101                         struct dentry *dentry, struct fileattr *fa);  
2102     int (*fileattr_get)(struct dentry *dentry, struct fileattr *fa);  
2103 } __cacheline_aligned;
```

```
const struct inode_operations ext4_file_inode_operations = {  
    .setattr      = ext4_setattr,  
    .getattr      = ext4_file_getattr,  
    .listxattr    = ext4_listxattr,  
    .get_acl     = ext4_get_acl,  
    .set_acl      = ext4_set_acl,  
    .fiemap       = ext4_fiemap,  
    .fileattr_get = ext4_fileattr_get,  
    .fileattr_set = ext4_fileattr_set,  
};
```

From: <https://elixir.bootlin.com/linux/latest/source/fs/ext4/file.c>

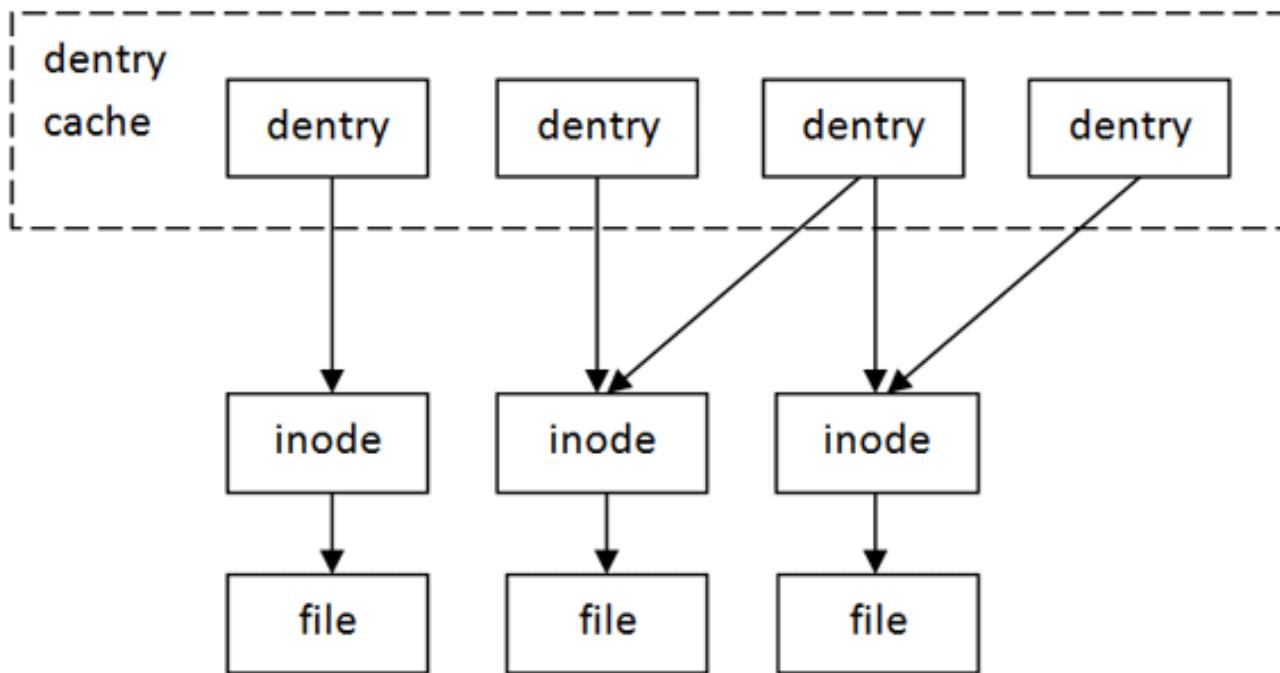
# Directory Entry - Dentry

As discussed, the VFS treats directories as files. An inode object represents both these components.

Despite this useful unification, the VFS often needs to perform directory-specific operations, such as path name lookup. To facilitate this, the VFS employs the concept of a directory entry (dentry).

Example: /bin/vi

- /
- bin
- vi



# Directory Entry - Dentry

The VFS constructs dentry objects on the fly, as needed, when performing directory operations.

Mainly target is mapping **File Name to Inode object**. Only the **partial** path name/component is stored in *d\_name*. If the name is short, it is stored in *d\_iname* to fast access.

```
90
91 struct dentry {
92     /* RCU Lookup touched fields */
93     unsigned int d_flags;           /* protected by d_lock */
94     seqcount_spinlock_t d_seq;    /* per dentry seqlock */
95     struct hlist_node d_hash;     /* Lookup hash List */
96     struct dentry *d_parent;      /* parent directory */
97     struct qstr d_name;
98     struct inode *d_inode;        /* Where the name belongs to - NULL is
99                                * negative */
100    unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
101
102    /* Ref Lookup also touches following */
103    struct lockref d_lockref;      /* per-dentry Lock and refcount */
104    const struct dentry_operations *d_op;
105    struct super_block *d_sb;      /* The root of the dentry tree */
106    unsigned long d_time;          /* used by d_revalidate */
107    void *d_fsdata;               /* fs-specific data */
108
109    union {
110        struct list_head d_lru;      /* LRU List */
111        wait_queue_head_t *d_wait;   /* in-Lookup ones only */
112    };
113    struct list_head d_child;      /* child of parent list */
114    struct list_head d_subdirs;    /* our children */
115
116    /* d_alias and d_rcu can share memory */
117    */
118    union {
119        struct hlist_node d_alias;   /* inode alias List */
120        struct hlist_node d_in_lookup_hash; /* only for in-Lookup ones */
121        struct rcu_head d_rcu;
122    } d_u;
123 } __randomize_layout;
```

```
74    /*
75     * Try to keep struct dentry aligned on 64 byte cachelines (this will
76     * give reasonable cacheline footprint with larger lines without the
77     * Large memory footprint increase).
78     */
79 #ifdef CONFIG_64BIT
80 # define DNAME_INLINE_LEN 32 /* 192 bytes */
81#else
82 # ifdef CONFIG_SMP
83 #  define DNAME_INLINE_LEN 36 /* 128 bytes */
84 # else
85 #  define DNAME_INLINE_LEN 40 /* 128 bytes */
86 # endif
87#endif
```

Dentry objects are represented by struct dentry and defined in <linux/dcache.h>.

# Directory Entry - Dentry

Kernel caches *dentry* objects in the *dentry* cache or, simply, the ***dcache***. A hash table and hashing function used to quickly resolve a given path into the associated *dentry* object.

All in memory active dentry objects are saved in a global hash list: *dentry\_hashtable*.

```
100 /*
101  * This is the single most critical data structure when it comes
102  * to the dcache: the hashtable for lookups. Somebody should try
103  * to make this good - I've just made it work.
104  *
105  * This hash-function tries to avoid losing too many bits of hash
106  * information, yet avoid using a prime hash-size or similar.
107  */
108
109 static unsigned int d_hash_shift __read_mostly;
110
111 static struct hlist_bl_head *dentry_hashtable __read_mostly;
112
113 static inline struct hlist_bl_head *d_hash(unsigned int hash)
114 {
115     return dentry_hashtable + (hash >> d_hash_shift);
116 }
```

<https://elixir.bootlin.com/linux/v5.13/source/fs/dcache.c#L101>

All in memory unused (*d\_count == 0*) dentry objects are saved in a filesystem instance global LRU list :*s\_dentry\_lru*.

```
1419 struct super_block {
1420     struct list_head      s_list;          /* Keep this first */
1421     dev_t                 s_dev;           /* search index; _not_ kdev_t */
1422     unsigned char          s_blocksize_bits;
1423     unsigned long           s_maxbytes;      /* Max file size */
1424     struct file_system_type *s_type;
1425     const struct super_operations  *s_op;
1426
1427
1428     /*
1429      * The list_lru structure is essentially just a pointer to a table
1430      * of per-node lru lists, each of which has its own spinlock.
1431      * There is no need to put them into separate cachelines.
1432      */
1433
1434     struct list_lru        s_dentry_lru;
1435     struct list_lru        s_inode_lru;
1436     struct rcu_head         rCU;
1437     struct work_struct     destroy_work;
1438 }
```

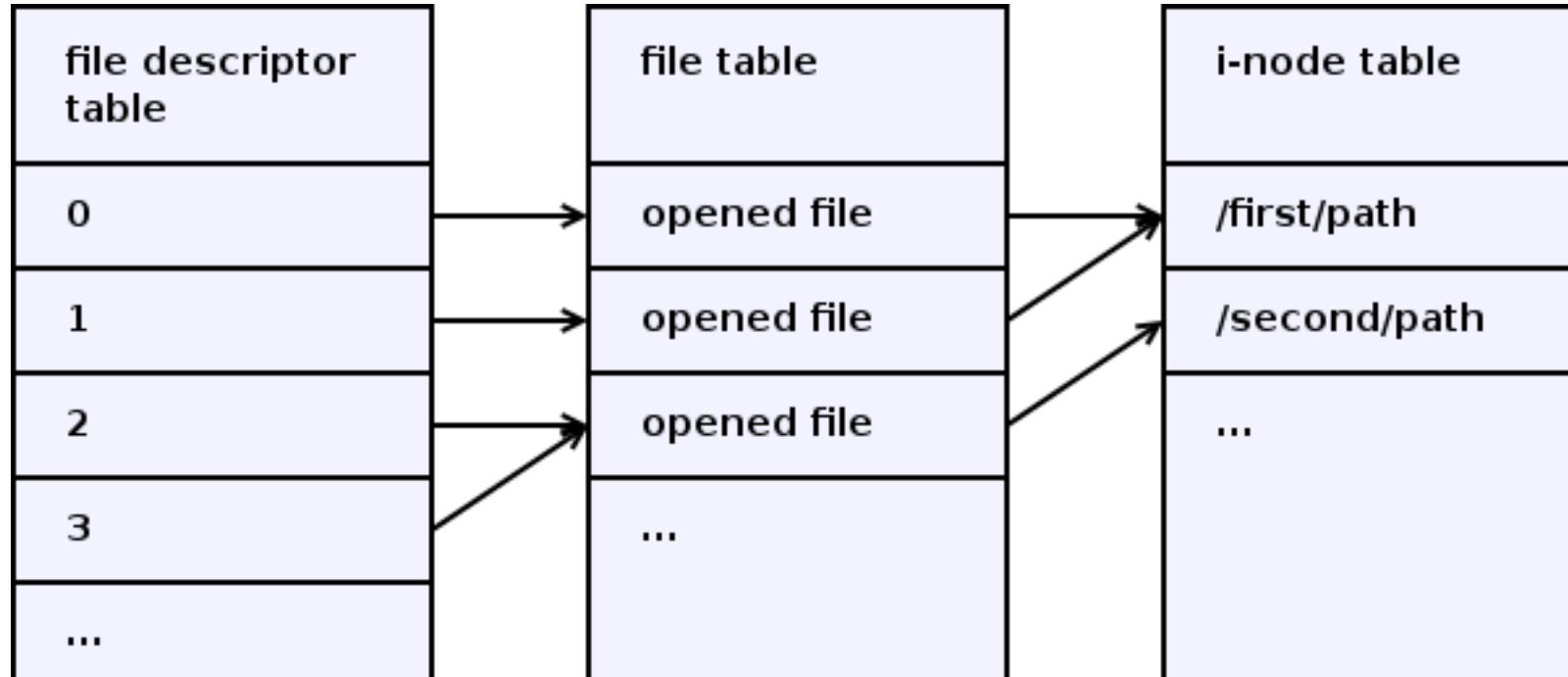
<https://elixir.bootlin.com/linux/v5.13/source/include/linux/fs.h#L1535>

# Directory Entry - Dentry

FS Specified dentry operations

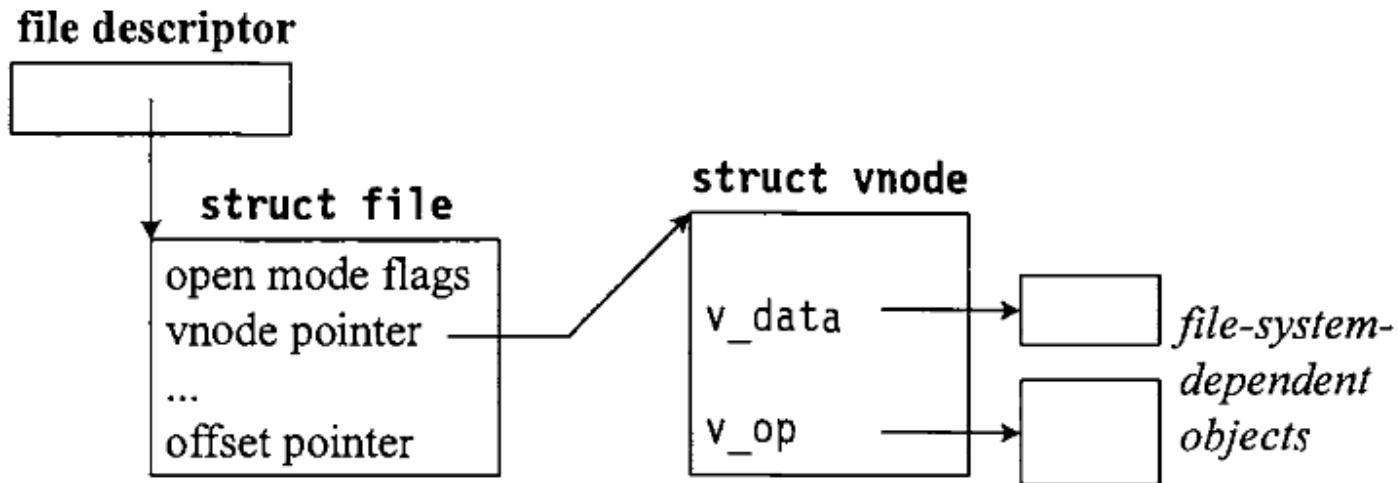
```
137 struct dentry_operations {
138     int (*d_revalidate)(struct dentry *, unsigned int);
139     int (*d_weak_revalidate)(struct dentry *, unsigned int);
140     int (*d_hash)(const struct dentry *, const struct qstr *);
141     int (*d_compare)(const struct dentry *,
142                      unsigned int, const char *, const struct qstr *);
143     int (*d_delete)(const struct dentry *);
144     int (*d_init)(struct dentry *);
145     void (*d_release)(struct dentry *);
146     void (*d_prune)(struct dentry *);
147     void (*d_iput)(struct dentry *, struct inode *);
148     char *(*d_dname)(struct dentry *, char *, int);
149     struct vfsmount *(*d_automount)(struct path *);
150     int (*d_manage)(const struct path *, bool);
151     struct dentry *(*d_real)(struct dentry *, const struct inode *);
152 } __cacheline_aligned;
```

# File Object



From: <https://gavv.github.io/articles/file-locks/>

# File Object



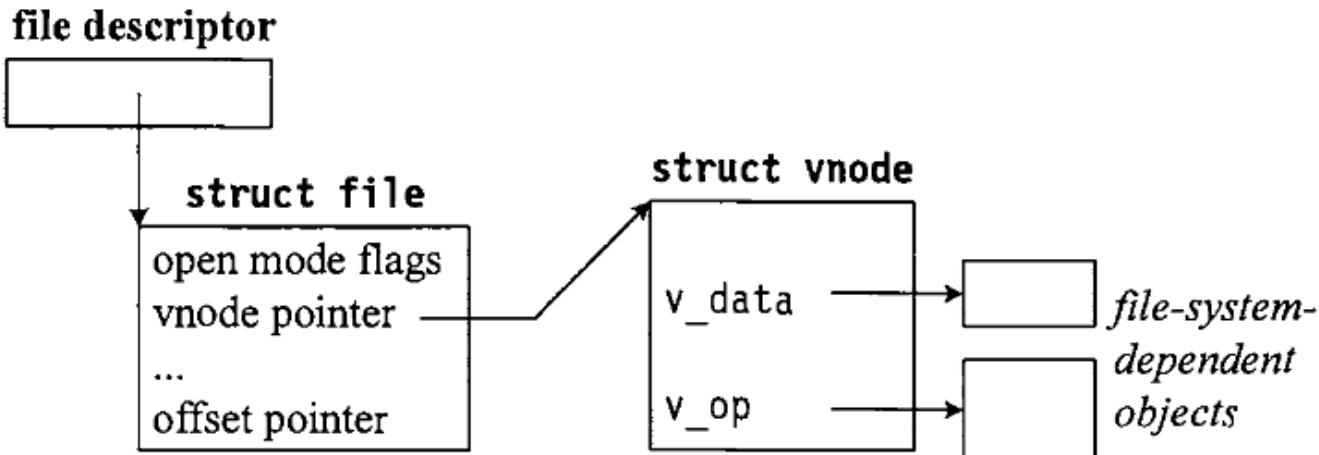
```
050
657 struct task_struct {
658 #ifdef CONFIG_THREAD_INFO_IN_TASK
659 /*
660  * For reasons of header soup (see current_thread_info()), this
661  * must be the first element of task_struct.
662  */
663 struct thread_info          thread_info;
664 #endif
665 /* -1 unrunnable, 0 runnable, >0 stopped: */
666 volatile long                state;
667
984
985
986
987
988
```

#include "fs.h"

*/\* Filesystem information: \*/*  
struct fs\_struct \*fs;

*/\* Open file information: \*/*  
struct files\_struct \*files;

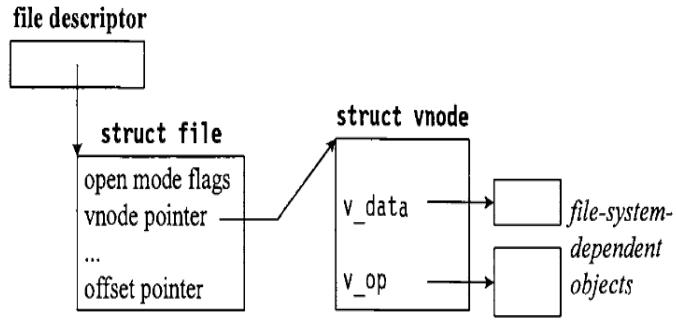
# File Object



```
/ include / linux / fdtable.h  
49 struct files_struct {  
50     /*  
51     * read mostly part  
52     */  
53     atomic_t count;  
54     bool resize_in_progress;  
55     wait_queue_head_t resize_wait;  
56  
57     struct fdtable __rcu *fdt;  
58     struct fdtable fdtab;  
59     /*  
60     * written part on a separate cache Line in SMP  
61     */  
62     spinlock_t file_lock __cacheline_aligned_in_smp;  
63     unsigned int next_fd;  
64     unsigned long close_on_exec_init[1];  
65     unsigned long open_fds_init[1];  
66     unsigned long full_fds_bits_init[1];  
67     struct file __rcu * fd_array[NR_OPEN_DEFAULT];  
68 };
```

```
/ include / linux / fdtable.h  
27 struct fdtable {  
28     unsigned int max_fds;  
29     struct file __rcu **fd;      /* current fd array */  
30     unsigned long *close_on_exec;  
31     unsigned long *open_fds;  
32     unsigned long *full_fds_bits;  
33     struct rcu_head rcu;  
34 };
```

# File Object



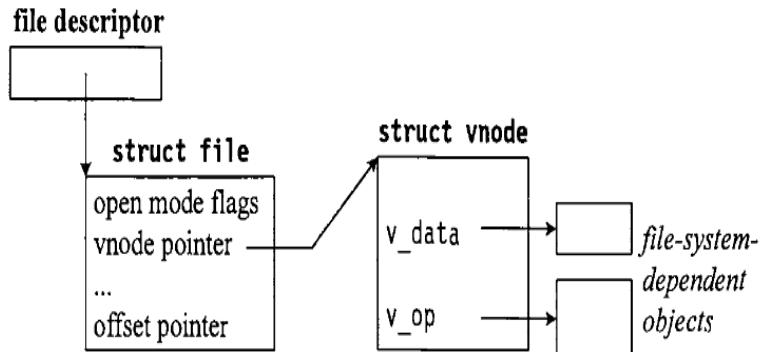
```
/ include / linux / fs.h

920 struct file {
921     union {
922         struct llist_node          fu_llist;
923         struct rcu_head            fu_rcuhead;
924     } f_u;
925     struct path                 f_path;
926     struct inode                *f_inode;           /* cached value */
927     const struct file_operations *f_op;

928     /*
929      * Protects f_ep, f_flags.
930      * Must not be taken from IRQ context.
931      */
932     spinlock_t                  f_lock;
933     enum rw_hint                f_write_hint;
934     atomic_long_t               f_count;
935     unsigned int                f_flags;
936     fmode_t                     f_mode;
937     struct mutex                f_pos_lock;
938     loff_t                      f_pos;
939     struct fown_struct          f_owner;
940     const struct cred            *f_cred;
941     struct file_ra_state        f_ra;

942     u64                         f_version;
943 #ifdef CONFIG_SECURITY
944     void                        *f_security;
945 #endif
946     /* needed for tty driver, and maybe others */
947     void                        *private_data;
948
949 #ifdef CONFIG_EPOLL
950     /* Used by fs/eventpoll.c to link all the hooks to this file */
951     struct hlist_head            *f_ep;
952 #endif /* #ifdef CONFIG_EPOLL */
953     struct address_space         *f_mapping;
954     errseq_t                     f_wb_err;
955     errseq_t                     f_sb_err; /* for syncfs */
956
957 } __randomize_layout
958 __attribute__((aligned(4))); /* Lest something weird decides that 2 is OK */
```

# File Object

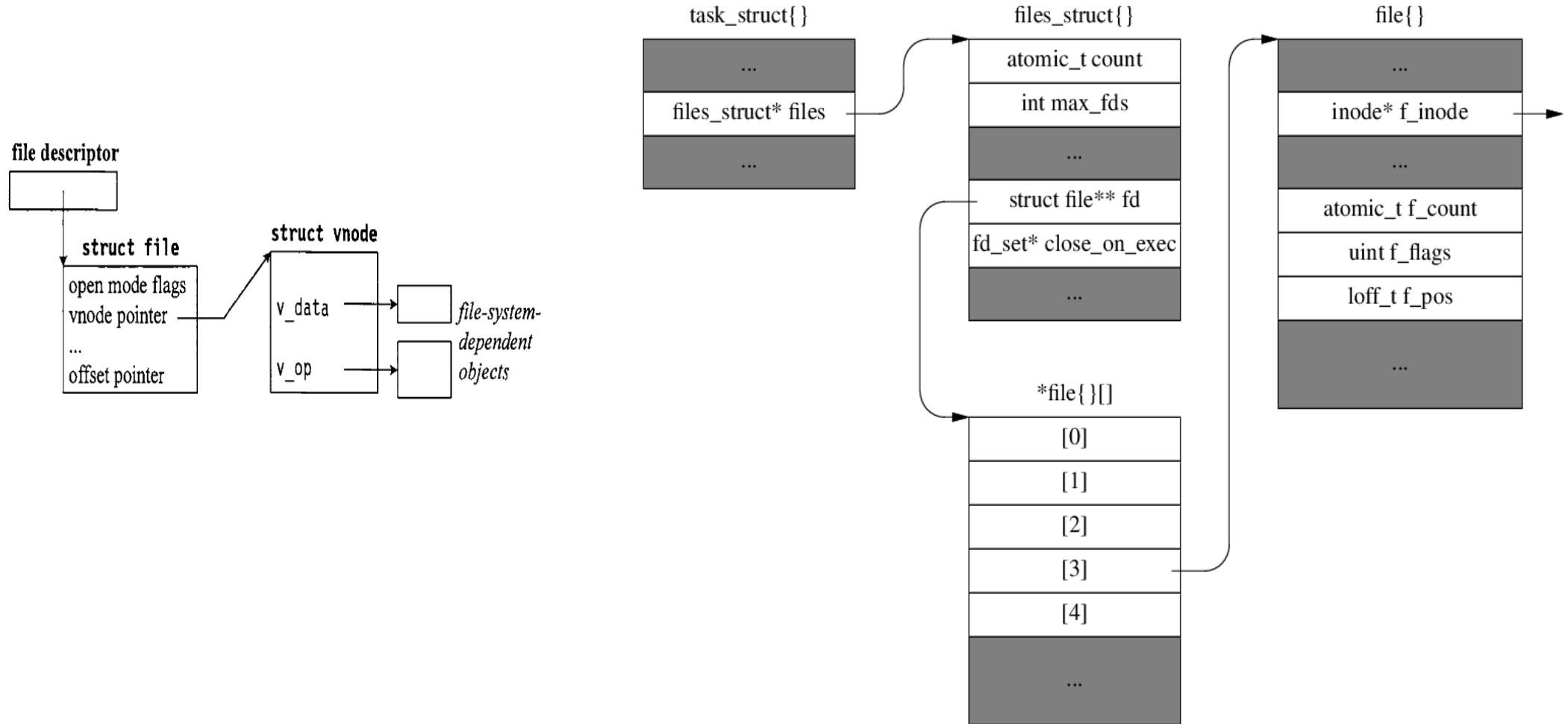


```
/ include / linux / fs.h
```

```
605  /*
606   * Keep mostly read-only and often accessed (especially for
607   * the RCU path lookup and 'stat' data) fields at the beginning
608   * of the 'struct inode'
609   */
610
611 struct inode {
612     umode_t             i_mode;
613     unsigned short      i_opflags;
614     kuid_t              i_uid;
615     kgid_t              i_gid;
616     unsigned int         i_flags;
617
618 #ifdef CONFIG_FS_POSIX_ACL
619     struct posix_acl    *i_acl;
620     struct posix_acl    *i_default_acl;
621 #endif
622
623     const struct inode_operations  *i_op;
624     struct super_block      *i_sb;
625     struct address_space    *i_mapping;
```

<https://elixir.bootlin.com/linux/v5.13/source/include/linux/fs.h#L611>

# File Object



From: <https://chenshuo.com/notes/kernel/fdt.png>

# File Object - VFS APIs & Ext4

```
2066
2069 struct file_operations {
2070     struct module *owner;
2071     loff_t (*llseek) (struct file *, loff_t, int);
2072     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
2073     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
2074     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
2075     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
2076     int (*iopoll)(struct kiocb *kiocb, struct io_comp_batch *,
2077                   unsigned int flags);
2078     int (*iterate) (struct file *, struct dir_context *);
2079     int (*iterate_shared) (struct file *, struct dir_context *);
2080     _poll_t (*poll) (struct file *, struct poll_table_struct *);
2081     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
2082     long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
2083     int (*mmap) (struct file *, struct vm_area_struct *);
2084     unsigned long mmap_supported_flags;
2085     int (*open) (struct inode *, struct file *);
2086     int (*flush) (struct file *, fl_owner_t id);
2087     int (*release) (struct inode *, struct file *);
2088     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
2089     int (*fasync) (int, struct file *, int);
2090     int (*lock) (struct file *, int, struct file_lock *);
2091     ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
2092     unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
2093     int (*check_flags)(int);
2094     int (*flock) (struct file *, int, struct file_lock *);
2095     ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
2096     ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
2097     int (*setlease)(struct file *, long, struct file_lock **, void **);
2098     long (*fallocate)(struct file *file, int mode, loff_t offset,
2099                       loff_t len);
2100     void (*show_fdinfo)(struct seq_file *m, struct file *f);
2101 #ifndef CONFIG_MMU
2102     unsigned (*mmap_capabilities)(struct file *);
2103 #endif
2104     ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
2105                               loff_t, size_t, unsigned int);
2106     loff_t (*remap_file_range)(struct file *file_in, loff_t pos_in,
2107                               struct file *file_out, loff_t pos_out,
2108                               loff_t len, unsigned int remap_flags);
2109     int (*fadvise)(struct file *, loff_t, loff_t, int);
2110 } __randomize_layout;
```

From: <https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L2069>

```
915 const struct file_operations ext4_file_operations = {
916     .llseek      = ext4_llseek,
917     .read_iter   = ext4_file_read_iter,
918     .write_iter  = ext4_file_write_iter,
919     .iopoll       = iocb_bio_iopoll,
920     .unlocked_ioctl = ext4_ioctl,
921 #ifdef CONFIG_COMPAT
922     .compat_ioctl = ext4_compat_ioctl,
923 #endif
924     .mmap        = ext4_file_mmap,
925     .mmap_supported_flags = MAP_SYNC,
926     .open         = ext4_file_open,
927     .release      = ext4_release_file,
928     .fsync        = ext4_sync_file,
929     .get_unmapped_area = thp_get_unmapped_area,
930     .splice_read  = generic_file_splice_read,
931     .splice_write = iter_file_splice_write,
932     .fallocate    = ext4_fallocate,
933 }
```

From: <https://elixir.bootlin.com/linux/latest/source/fs/ext4/file.c>

# Debugfs – Prowl around a FS

DEBUGFS(8)	System Manager's Manual	DEBUGFS(8)
<b>NAME</b> debugfs - ext2/ext3/ext4 file system debugger		
<b>SYNOPSIS</b> <code>debugfs [ -DVwcin ] [ -b blocksize ] [ -s superblock ] [ -f cmd_file ] [ -R request ] [ -d data_source_device ] [ -z undo_file ] [ device ]</code>		
<b>DESCRIPTION</b> The <b>debugfs</b> program is an interactive file system debugger. It can be used to examine and change the state of an ext2, ext3, or ext4 file system.  <u>device</u> is a block device (e.g., <code>/dev/sdXX</code> ) or a file containing the file system.		
<b>OPTIONS</b> <b>-w</b> Specifies that the file system should be opened in read-write mode. Without this option, the file system is opened in read-only mode.  <b>-n</b> Disables metadata checksum verification. This should only be used if you believe the metadata to be correct despite the complaints of <code>e2fsprogs</code> .  <b>-c</b> Specifies that the file system should be opened in catastrophic mode, in which the inode and group bitmaps are not read initially. This can be useful for filesystems with significant corruption, but because of this, catastrophic mode forces the filesystem to be opened read-only.  <b>-i</b> Specifies that <u>device</u> represents an ext2 image file created by the <code>e2image</code> program. Since the ext2 image file only contains the superblock, block group descriptor, block and inode allocation bitmaps, and the inode table, many <b>debugfs</b> commands will not function properly. <b>Warning:</b> no safety checks are in place, and <b>debugfs</b> may fail in interesting ways if commands such as <code>ls</code> , <code>dump</code> , etc. are tried without specifying the <u>data_source_device</u> using the <u>-d</u> option. <b>debugfs</b> is a debugging tool. It has rough edges!  <b>-d data_source_device</b> Used with the <u>-i</u> option, specifies that <u>data_source_device</u> should be used when reading blocks not found in the ext2 image file. This includes data, directory, and indirect blocks.  <b>-b blocksize</b> Forces the use of the given block size (in bytes) for the file system, rather than detecting the correct block size automatically. (This option is rarely needed; it is used primarily when the file system is extremely badly damaged/corrupted.)  <b>-s superblock</b> Causes the file system superblock to be read from the given block number, instead of using the primary superblock (located at an offset of 1024 bytes from the beginning of the filesystem). If you specify the <u>-s</u> option, you must also provide the blocksize of the filesystem via the <u>-b</u> option. (This option is rarely needed; it is used primarily when the file system is extremely badly damaged/corrupted.)		

# Debugfs – Prowl around a FS

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ debugfs mydisk
debugfs 1.45.5 (07-Jan-2020)
debugfs: stat /home/zegang/1.txt
```

```
Inode: 14  Type: regular  Mode: 0664  Flags: 0x80000
Generation: 982170681  Version: 0x00000001
User: 1000  Group: 1000  Size: 2
File ACL: 0
Links: 1  Blockcount: 8
Fragment: Address: 0  Number: 0  Size: 0
ctime: 0x6217565c -- Thu Feb 24 17:56:44 2022
atime: 0x62175665 -- Thu Feb 24 17:56:53 2022
mtime: 0x6217565c -- Thu Feb 24 17:56:44 2022
Inode checksum: 0x0000c357
EXTENTS:
(0):5147
(END)
```

# Debugfs – Inode

```
/*
 * Structure of an inode on the disk
 */
struct ext4_inode {
    __le16 i_mode;          /* File mode */
    __le16 i_uid;           /* Low 16 bits of Owner Uid */
    __le32 i_size_lo;       /* Size in bytes */
    __le32 i_atime;         /* Access time */
    __le32 i_ctime;         /* Inode Change time */
    __le32 i_mtime;         /* Modification time */
    __le32 i_dtime;         /* Deletion Time */
    __le16 i_gid;           /* Low 16 bits of Group Id */
    __le16 i_links_count;   /* Links count */
    __le32 i_blocks_lo;     /* Blocks count */
    __le32 i_flags;         /* File flags */
    union {
```

```
debugfs: id /home/zegang/1.txt
0000 b481 e803 0200 0000 6556 1762 5c56 1762 .....eV.b\V.b
0020 5c56 1762 0000 0000 e803 0100 0800 0000 \V.b.....
0040 0000 0800 0100 0000 0af3 0100 0400 0000 .....
0060 0000 0000 0000 0000 0100 0000 1b14 0000 .....
0100 0000 0000 0000 0000 0000 0000 0000 0000 .....
*
0140 0000 0000 39bc 8a3a 0000 0000 0000 0000 ....9.....
0160 0000 0000 0000 0000 0000 0000 57c3 0000 .....W...
```

```
Inode: 14 Type: regular Mode: 0664 Flags: 0x800000
Generation: 982170681 Version: 0x00000001
User: 1000 Group: 1000 Size: 2
File ACL: 0
Links: 1 Blockcount: 8
Fragment: Address: 0 Number: 0 Size: 0
ctime: 0x6217565c -- Thu Feb 24 17:56:44 2022
atime: 0x62175665 -- Thu Feb 24 17:56:53 2022
mtime: 0x6217565c -- Thu Feb 24 17:56:44 2022
Inode checksum: 0x0000c357
EXTENTS:
(0):5147
(END)
```

```
zegang@zegang-VirtualBox:~/1300-simple-fs$ python3 -c "print(hex(1000))"
0x3e8
zegang@zegang-VirtualBox:~/1300-simple-fs$ python3 -c "print(hex(0o664))"
0x1b4
```

# Syscalls

```
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp/home/zegang$ strace -e trace=file ls -lh 1.txt
execve("/usr/bin/ls", ["ls", "-lh", "1.txt"], 0x7ffc60509530 /* 56 vars */) = 0
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre2-8.so.0", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
statfs("/sys/fs/selinux", 0x7ffe2bd3cb00) = -1 ENOENT (No such file or directory)
statfs("/selinux", 0x7ffe2bd3cb00)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/proc/filesystems", O_RDONLY|O_CLOEXEC) = 3
access("/etc/selinux/config", F_OK)     = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/share/locale/locale.alias", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/share/locale/zh_CN.UTF-8/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/zh_CN.utf8/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/zh_CN/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/zh.UTF-8/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/zh.utf8/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/zh/LC_TIME/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache", O_RDONLY) = 3
lstat("1.txt", {st_mode=S_IFREG|0664, st_size=2, ...}) = 0
lgetxattr("1.txt", "security.selinux", 0x55af7a7be390, 255) = -1 ENODATA (No data available)
getxattr("1.txt", "system.posix_acl_access", NULL, 0) = -1 ENODATA (No data available)
openat(AT_FDCWD, "/etc/nsswitch.conf", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libnss_files.so.2", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/passwd", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/group", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
-rw-rw-r-- 1 zegang zegang 2 月 24 17:56 1.txt
+++ exited with 0 +++
```

# Syscalls: POSIX

**INDEX**

Istat  
Search..

[Alphabetic | Topic | Word Search]

Select a Volume:  
[Base Definitions | System Interfaces | Shell & Utilities | Rationale]  
[Frontmatter]  
[Main Index]

---

**Base Definitions**

- 1. Introduction
- 2. Conformance
- 3. Definitions
- 4. General Concepts
- 5. File Format Notation
- 6. Character Set
- 7. Locale
- 8. Environment Variables
- 9. Regular Expressions
- 10. Directory Structure and Devices
- 11. General Terminal Interface
- 12. Utility Conventions
- 13. Headers

<< Previous

Home

The Open Group Base Specifications Issue 7, 2018 edition  
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)  
Copyright © 2001-2018 IEEE and The Open Group

---

**NAME**

fstatat, Istat, stat - get file status

**SYNOPSIS**

```
[@] #include <fcntl.h>
#include <sys/stat.h>

int fstatat(int fd, const char *restrict path,
            struct stat *restrict buf, int flag);
int lstat(const char *restrict path, struct stat *restrict buf);
int stat(const char *restrict path, struct stat *restrict buf);
```

**DESCRIPTION**

The `stat()` function shall obtain information about the named file and write it to the area pointed to by the `buf` argument. The `path` argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause `stat()` to fail. In particular, the system may deny the existence of the file specified by `path`.

If the named file is a symbolic link, the `stat()` function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The `buf` argument is a pointer to a `stat` structure, as defined in the `<sys/stat.h>` header, into which information is placed concerning the file.

The `stat()` function shall update any time-related fields (as described in XBD [File Times Update](#)), before writing into the `stat` structure.

[SHM] If the named file is a shared memory object, the implementation shall update in the `stat` structure pointed to by the `buf` argument the `st_uid`, `st_gid`, `st_size`, and `st_mode` fields, and only the `S_IRUSR`, `S_IWUSR`, `S_IRGRP`, `S_IWGRP`, `S_IROTH`, and `S_IWOTH` file permission bits need be valid. The implementation may update other fields and flags. ☐

[TVM] If the named file is a typed memory object, the implementation shall update in the `stat` structure pointed to by the `buf` argument the `st_uid`, `st_gid`, `st_size`, and `st_mode` fields, and only the `S_IRUSR`, `S_IWUSR`, `S_IRGRP`, `S_IWGRP`, `S_IROTH`, and `S_IWOTH` file permission bits need be valid. The implementation may update other fields and flags. ☐

For all other file types defined in this volume of POSIX.1-2017, the structure members `st_mode`, `st_ino`, `st_dev`, `st_uid`, `st_gid`, `st_atim`, `st_ctim`, and `st_mtum` shall have meaningful values and the value of the member `st_nlink` shall be set to the number of links to the file.

The `Istat()` function shall be equivalent to `stat()`, except when `path` refers to a symbolic link. In that case `Istat()` shall return information about the link, while `stat()` shall return information about the file the link references.

For symbolic links, the `st_mode` member shall contain meaningful information when used with the file type macros. The file mode bits in `st_mode` are unspecified. The structure members `st_ino`, `st_dev`, `st_uid`, `st_gid`, `st_atim`, `st_ctim`, and `st_mtum` shall have meaningful values and the value of the `st_nlink` member shall be set to the number of (hard) links to the symbolic link. The value of the `st_size` member shall be set to the length of the pathname contained in the symbolic link not including any terminating null byte.

The `fstatat()` function shall be equivalent to the `stat()` or `Istat()` function, depending on the value of `flag` (see below), except in the case where `path` specifies a relative path. In this case the status shall be retrieved from a file relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not `O_SEARCH`, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is `O_SEARCH`, the function shall not perform the check.

Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

AT\_SYMLINK\_NOFOLLOW  
If `path` names a symbolic link, the status of the symbolic link is returned.

If `fstatat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working directory shall be used and the behavior shall be identical to a call to `stat()` or `Istat()` respectively, depending on whether or not the `AT_SYMLINK_NOFOLLOW` bit is set in `flag`.

**RETURN VALUE**

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set `errno` to indicate the error.

**ERRORS**

These functions shall fail if:

[EACCES]  
Search permission is denied for a component of the path prefix.

[EIO]  
An error occurred while reading from the file system.

[ELOOP]  
A loop exists in symbolic links encountered during resolution of the `path` argument.

[ENAMETOOLONG]  
The length of a component of a pathname is longer than `{NAME_MAX}`.

[ENOENT]  
A component of `path` does not name an existing file or `path` is an empty string.

[ENOTDIR]  
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the `path` argument contains at least one non- <slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

# Syscalls

```
zegang@zegang-VirtualBox:~/1300-simple-fs/mydisk_mp/home/zegang$ man 2 lstat | head -15
STAT(2)                                              Linux Programmer's Manual
STAT(2)

NAME
    stat, fstat, lstat, fstatat - get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *pathname, struct stat *statbuf);
int fstat(int fd, struct stat *statbuf);
int lstat(const char *pathname, struct stat *statbuf);

#include <fcntl.h>          /* Definition of AT_* constants */
```

```
1230
1239 /* obsolete: fs/stat.c */
1240 asmlinkage long sys_stat(const char __user *filename,
1241                         struct __old_kernel_stat __user *statbuf);
1242 asmlinkage long sys_lstat(const char __user *filename,
1243                           struct __old_kernel_stat __user *statbuf);
1244 asmlinkage long sys_fstat(unsigned int fd,
1245                           struct __old_kernel_stat __user *statbuf);
1246 asmlinkage long sys_readlink(const char __user *path,
1247                             char __user *buf, int bufsiz);
```

<https://elixir.bootlin.com/linux/latest/source/include/linux/syscalls.h#L1242>

```
302
303 SYSCALL_DEFINE2(lstat, const char __user *, filename,
304                  struct __old_kernel_stat __user *, statbuf)
305 {
306     struct kstat stat;
307     int error;
308
309     error = vfs_lstat(filename, &stat);
310     if (error)
311         return error;
312
313     return cp_old_stat(&stat, statbuf);
314 }
315
```

<https://elixir.bootlin.com/linux/latest/source/fs/stat.c#L303>

# Syscalls

```
5554 }
5555 static inline int vfs_lstat(const char __user *name, struct kstat *stat)
5556 {
5557     return vfs_fstatat(AT_FDCWD, name, stat, AT_SYMLINK_NOFOLLOW);
5558 }
```

<https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L3355>

```
239
240 int vfs_fstatat(int dfd, const char __user *filename,
241                 struct kstat *stat, int flags)
242 {
243     return vfs_statx(dfd, filename, flags | AT_NO_AUTOMOUNT,
244                       stat, STATX_BASIC_STATS);
245 }
```

<https://elixir.bootlin.com/linux/latest/source/fs/stat.c#L240>

```
92     */
93     int vfs_getattr_nosec(const struct path *path, struct kstat *stat,
94                           u32 request_mask, unsigned int query_flags)
95     {
96         struct user_namespace *mnt_userns;
97         struct inode *inode = d_backing_inode(path->dentry);
98
99         memset(stat, 0, sizeof(*stat));
100        stat->result_mask |= STATX_BASIC_STATS;
101        query_flags &= AT_STATX_SYNC_TYPE;
102
103        /* allow the fs to override these if it really wants to */
104        /* SB_NOATIME means filesystem supplies dummy atime value */
105        if (inode->i_sb->s_flags & SB_NOATIME)
106            stat->result_mask &= ~STATX_ATIME;
107
108        /*
109         * Note: If you add another clause to set an attribute flag, please
110         * update attributes_mask below.
111         */
112        if (IS_AUTOMOUNT(inode))
113            stat->attributes |= STATX_ATTR_AUTOMOUNT;
114
115        if (IS_DAX(inode))
116            stat->attributes |= STATX_ATTR_DAX;
117
118        stat->attributes_mask |= (STATX_ATTR_AUTOMOUNT |
119                                  STATX_ATTR_DAX);
120
121        mnt_userns = mnt_user_ns(path->mnt);
122        if (inode->i_op->getattr)
123            return inode->i_op->getattr(mnt_userns, path, stat,
124                                         request_mask, query_flags);
125
126        generic_fillattr(mnt_userns, inode, stat);
127
128    }
```

<https://elixir.bootlin.com/linux/latest/source/fs/stat.c#L123>

# References

- <https://en.wikipedia.org/wiki/Ext4>
- <https://phoenixnap.com/kb/linux-file-system>
- <http://www.linfo.org/superblock>
- <https://elixir.bootlin.com/linux/latest/source/fs/ext4/file.c>
- <https://elixir.bootlin.com/linux/latest/source/fs/stat.c#L123>

# Homework

1. Why is there only one Block Group in the mydisk ext4 format ?
2. Could you make a format disk which has more than 1 block groups ?
3. Why are block numbers of block/inode bitmap 16 instead of 1 ?
4. Starting in ext4, there is a new feature called flexible block groups. How ext4 disk format if this feature enabled ?
5. Write a file with content “Hello 1300-simple-fs” and hexdump the file data on disk.
6. Show the data on disk layout of a big file.

```
zegang@zegang-VirtualBox:~$ mkdir 1300-simple-fs
zegang@zegang-VirtualBox:~$ cd 1300-simple-fs/
zegang@zegang-VirtualBox:~/1300-simple-fs$ touch mydisk
zegang@zegang-VirtualBox:~/1300-simple-fs$ truncate -s 128M mydisk
zegang@zegang-VirtualBox:~/1300-simple-fs$ ll -h mydisk
-rw-rw-r-- 1 zegang zegang 128M 2月 24 17:33 mydisk
```

Reference Paper:

1. Kevin D. Fairbanks, An analysis of Ext4 for digital forensics, Digital Investigation, Volume 9, Supplement, 2012, Pages S118-S130, ISSN 1742-2876,  
<https://doi.org/10.1016/j.dlin.2012.05.010>.(<https://www.sciencedirect.com/science/article/pii/S1742287612000357>)

# Homework

## Background

### Ten Inodes FS (TIF)

This demo FUSE filesystem is just a easy start point to felling some basic VFS objects.

This FS is designed for only limited number of files/inodes scenario, 10.

## HOW TO RUN

After install libfuse librarys, run "./test.sh".

## TAKE IT AWAY

Ops, mkdir/ls, on the mounted FS and check logs to watch VFS objects create/filling...

This demo FUSE filesystem is in-memory only and some implementation differes from VFS standars. Such as, dentry, dcache. You may change it for yourself.

## Homework

1. Allocated 1 data block for 1 file.
2. Write data into the file data block.
3. Bind the file data block with file inode.
4. Any what your want...

```
zegang@zegang-VirtualBox:~/github/fuse/tenindoesfs$ ./test.sh
umount: mymnt: no mount point specified.
gcc -g -o fuse tenindoesfs main.c -D_FILE_OFFSET_BITS=64 -lfuse
  File: "mymnt"
    ID: 0      Namelen: 255      Type: fuseblk
  Block size: 1024      Fundamental block size: 1024
  Blocks: Total: 10      Free: 10      Available: 10
  Inodes: Total: 10      Free: 10

-----To mkdir mymnt/d1
total 1
2 drwxrwxr-x 1 root root 0 1月 1 1970 d1

-----To check FS stat again
  File: "mymnt"
    ID: 0      Namelen: 255      Type: fuseblk
  Block size: 1024      Fundamental block size: 1024
  Blocks: Total: 10      Free: 10      Available: 10
  Inodes: Total: 10      Free: 9
zegang@zegang-VirtualBox:~/github/fuse/tenindoesfs$ for i in {0..8}; do touch mymnt/f${i}; done
touch: setting times of 'mymnt/f0': Function not implemented
touch: setting times of 'mymnt/f1': Function not implemented
touch: setting times of 'mymnt/f2': Function not implemented
touch: setting times of 'mymnt/f3': Function not implemented
touch: setting times of 'mymnt/f4': Function not implemented
touch: setting times of 'mymnt/f5': Function not implemented
touch: setting times of 'mymnt/f6': Function not implemented
touch: setting times of 'mymnt/f7': Function not implemented
touch: setting times of 'mymnt/f8': Function not implemented
zegang@zegang-VirtualBox:~/github/fuse/tenindoesfs$ ls -il mymnt
total 5
2 drwxrwxr-x 1 root root 0 1月 1 1970 d1
3 -rw-rw-r-- 1 root root 0 1月 1 1970 f0
4 -rw-rw-r-- 1 root root 0 1月 1 1970 f1
5 -rw-rw-r-- 1 root root 0 1月 1 1970 f2
6 -rw-rw-r-- 1 root root 0 1月 1 1970 f3
7 -rw-rw-r-- 1 root root 0 1月 1 1970 f4
8 -rw-rw-r-- 1 root root 0 1月 1 1970 f5
9 -rw-rw-r-- 1 root root 0 1月 1 1970 f6
10 -rw-rw-r-- 1 root root 0 1月 1 1970 f7
11 -rw-rw-r-- 1 root root 0 1月 1 1970 f8
zegang@zegang-VirtualBox:~/github/fuse/tenindoesfs$ touch mymnt/f9
touch: setting times of 'mymnt/f9': No such file or directory
```