Linux structs (v 5.17.5)

Contents

1 File system structs

1.1 file_system_type

```
struct file_system_type {
    const char *name;
    int fs_flags;
#define FS_REQUIRES_DEV         1
#define FS_BINARY_MOUNTDATA     2
#define FS_HAS_SUBTYPE          4
#define FS_USERNS_MOUNT         8       /* Can be mounted by userns root */
#define FS_DISALLOW_NOTIFY_PERM 16      /* Disable fanotify permission events */
#define FS_ALLOW_IDMAP         32      /* FS has been updated to handle vfs idmappings.
    */
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename()
    internally. */
    int (*init_fs_context)(struct fs_context *);
    const struct fs_parameter_spec *parameters;
    struct dentry *(*mount) (struct file_system_type *, int,
                const char *, void *);
    void (*kill_sb) (struct super_block *);
    struct module *owner;
    struct file_system_type * next;
    struct hlist_head fs_supers;

    struct lock_class_key s_lock_key;
    struct lock_class_key s_umount_key;
    struct lock_class_key s_vfs_rename_key;
    struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];

    struct lock_class_key i_lock_key;
    struct lock_class_key i_mutex_key;
    struct lock_class_key invalidate_lock_key;
    struct lock_class_key i_mutex_dir_key;
};
```

1.2 vfsmount

```
#define MNT_NOSUID          0x01
#define MNT_NODEV           0x02
#define MNT_NOEXEC          0x04
#define MNT_NOATIME         0x08
#define MNT_NODIRATIME  0x10
#define MNT_RELATIME    0x20
#define MNT_READONLY    0x40    /* does the user want this to be r/o? */
#define MNT_NOSYMFOLLOW 0x80

#define MNT_SHRINKABLE  0x100
#define MNT_WRITE_HOLD  0x200

#define MNT_SHARED          0x1000      /* if the vfsmount is a shared mount */
#define MNT_UNBINDABLE  0x2000  /* if the vfsmount is a unbindable mount */
/*
 * MNT_SHARED_MASK is the set of flags that should be cleared when a
 * mount becomes shared.  Currently, this is only the flag that says a
 * mount cannot be bind mounted, since this is how we create a mount
 * that shares events with another mount.  If you add a new MNT_*
 * flag, consider how it interacts with shared mounts.
 */
#define MNT_SHARED_MASK         (MNT_UNBINDABLE)
#define MNT_USER_SETTABLE_MASK  (MNT_NOSUID | MNT_NODEV | MNT_NOEXEC \
                                                | MNT_NOATIME | MNT_NODIRATIME |
                                                    MNT_RELATIME \
                                                | MNT_READONLY | MNT_NOSYMFOLLOW)
#define MNT_ATIME_MASK          (MNT_NOATIME | MNT_NODIRATIME | MNT_RELATIME )
```

```
#define MNT_INTERNAL_FLAGS      (MNT_SHARED | MNT_WRITE_HOLD | MNT_INTERNAL | \
                                 MNT_DOOMED | MNT_SYNC_UMOUNT | MNT_MARKED | \
                                 MNT_CURSOR)

#define MNT_INTERNAL        0x4000

#define MNT_LOCK_ATIME          0x040000
#define MNT_LOCK_NOEXEC         0x080000
#define MNT_LOCK_NOSUID         0x100000
#define MNT_LOCK_NODEV          0x200000
#define MNT_LOCK_READONLY       0x400000
#define MNT_LOCKED                  0x800000
#define MNT_DOOMED                  0x1000000
#define MNT_SYNC_UMOUNT         0x2000000
#define MNT_MARKED                  0x4000000
#define MNT_UMOUNT                  0x8000000
#define MNT_CURSOR                  0x10000000

struct vfsmount {
        struct dentry *mnt_root;        /* root of the mounted tree */
        struct super_block *mnt_sb;     /* pointer to superblock */
        int mnt_flags;
        struct user_namespace *mnt_userns;
} __randomize_layout;
```

1.3 super_block

```
struct super_block {
    struct list_head    s_list;         /* Keep this first */
    dev_t                       s_dev;          /* search index; _not_ kdev_t */
    unsigned char               s_blocksize_bits;
    unsigned long               s_blocksize;
    loff_t                      s_maxbytes;     /* Max file size */
    struct file_system_type     *s_type;
    const struct super_operations       *s_op;
    const struct dquot_operations       *dq_op;
    const struct quotactl_ops   *s_qcop;
    const struct export_operations *s_export_op;
    unsigned long               s_flags;
    unsigned long               s_iflags;       /* internal SB_I_* flags */
    unsigned long               s_magic;
    struct dentry               *s_root;
    struct rw_semaphore s_umount;
    int                 s_count;
    atomic_t            s_active;
#ifdef CONFIG_SECURITY
    void                    *s_security;
#endif
    const struct xattr_handler **s_xattr;
#ifdef CONFIG_FS_ENCRYPTION
    const struct fscrypt_operations     *s_cop;
    struct key          *s_master_keys; /* master crypto keys in use */
#endif
#ifdef CONFIG_FS_VERITY
    const struct fsverity_operations *s_vop;
#endif
#if IS_ENABLED(CONFIG_UNICODE)
    struct unicode_map *s_encoding;
    __u16 s_encoding_flags;
#endif
    struct hlist_bl_head        s_roots;        /* alternate root dentries for NFS */
    struct list_head    s_mounts;       /* list of mounts; _not_ for fs use */
    struct block_device *s_bdev;
    struct backing_dev_info *s_bdi;
```

```
    struct mtd_info             *s_mtd;
    struct hlist_node   s_instances;
    unsigned int                s_quota_types;  /* Bitmask of supported quota types */
    struct quota_info   s_dquot;        /* Diskquota specific options */

    struct sb_writers   s_writers;

    /*
        * Keep s_fs_info, s_time_gran, s_fsnotify_mask, and
        * s_fsnotify_marks together for cache efficiency. They are frequently
        * accessed and rarely modified.
        */
    void                        *s_fs_info;     /* Filesystem private info */

    /* Granularity of c/m/atime in ns (cannot be worse than a second) */
    u32                 s_time_gran;
    /* Time limits for c/m/atime in seconds */
    time64_t                s_time_min;
    time64_t                s_time_max;
#ifdef CONFIG_FSNOTIFY
    __u32                       s_fsnotify_mask;
    struct fsnotify_mark_connector __rcu        *s_fsnotify_marks;
#endif

    char                        s_id[32];       /* Informational name */
    uuid_t                      s_uuid;         /* UUID */

    unsigned int                s_max_links;
    fmode_t                     s_mode;

    /*
        * The next field is for VFS *only*. No filesystems have any business
        * even looking at it. You had been warned.
        */
    struct mutex s_vfs_rename_mutex;    /* Kludge */

    /*
        * Filesystem subtype.  If non-empty the filesystem type field
        * in /proc/mounts will be "type.subtype"
        */
    const char *s_subtype;

    const struct dentry_operations *s_d_op; /* default d_op for dentries */

    struct shrinker s_shrink;   /* per-sb shrinker handle */

    /* Number of inodes with nlink == 0 but still referenced */
    atomic_long_t s_remove_count;

    /*
        * Number of inode/mount/sb objects that are being watched, note that
        * inodes objects are currently double-accounted.
        */
    atomic_long_t s_fsnotify_connectors;

    /* Being remounted read-only */
    int s_readonly_remount;

    /* per-sb errseq_t for reporting writeback errors via syncfs */
    errseq_t s_wb_err;

    /* AIO completions deferred from interrupt context */
    struct workqueue_struct *s_dio_done_wq;
    struct hlist_head s_pins;

    /*
```

```
        * Owning user namespace and default context in which to
        * interpret filesystem uids, gids, quotas, device nodes,
        * xattrs and security labels.
        */
    struct user_namespace *s_user_ns;

    /*
        * The list_lru structure is essentially just a pointer to a table
        * of per-node lru lists, each of which has its own spinlock.
        * There is no need to put them into separate cachelines.
        */
    struct list_lru             s_dentry_lru;
    struct list_lru             s_inode_lru;
    struct rcu_head             rcu;
    struct work_struct  destroy_work;

    struct mutex                s_sync_lock;    /* sync serialisation lock */

    /*
        * Indicates how deep in a filesystem stack this SB is
        */
    int s_stack_depth;

    /* s_inode_list_lock protects s_inodes */
    spinlock_t          s_inode_list_lock ____cacheline_aligned_in_smp;
    struct list_head    s_inodes;       /* all inodes */

    spinlock_t          s_inode_wblist_lock;
    struct list_head    s_inodes_wb;    /* writeback inodes */
} __randomize_layout;
```

1.4 super_operations

```
struct super_operations {
        struct inode *(*alloc_inode)(struct super_block *sb);
        void (*destroy_inode)(struct inode *);
        void (*free_inode)(struct inode *);

        void (*dirty_inode) (struct inode *, int flags);
        int (*write_inode) (struct inode *, struct writeback_control *wbc);
        int (*drop_inode) (struct inode *);
        void (*evict_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        int (*sync_fs)(struct super_block *sb, int wait);
        int (*freeze_super) (struct super_block *);
        int (*freeze_fs) (struct super_block *);
        int (*thaw_super) (struct super_block *);
        int (*unfreeze_fs) (struct super_block *);
        int (*statfs) (struct dentry *, struct kstatfs *);
        int (*remount_fs) (struct super_block *, int *, char *);
        void (*umount_begin) (struct super_block *);

        int (*show_options)(struct seq_file *, struct dentry *);
        int (*show_devname)(struct seq_file *, struct dentry *);
        int (*show_path)(struct seq_file *, struct dentry *);
        int (*show_stats)(struct seq_file *, struct dentry *);
#ifdef CONFIG_QUOTA
        ssize_t (*quota_read)(struct super_block *, int, char *, size_t, loff_t);
        ssize_t (*quota_write)(struct super_block *, int, const char *, size_t, loff_t);
        struct dquot **(*get_dquots)(struct inode *);
#endif
        long (*nr_cached_objects)(struct super_block *,
                                  struct shrink_control *);
        long (*free_cached_objects)(struct super_block *,
                                    struct shrink_control *);
```

```
};
```

1.5 inode

```
/*
 * Keep mostly read-only and often accessed (especially for
 * the RCU path lookup and 'stat' data) fields at the beginning
 * of the 'struct inode'
 */
struct inode {
    umode_t                     i_mode;
    unsigned short              i_opflags;
    kuid_t                      i_uid;
    kgid_t                      i_gid;
    unsigned int                i_flags;

#ifdef CONFIG_FS_POSIX_ACL
    struct posix_acl    *i_acl;
    struct posix_acl    *i_default_acl;
#endif

    const struct inode_operations       *i_op;
    struct super_block  *i_sb;
    struct address_space        *i_mapping;

#ifdef CONFIG_SECURITY
    void                        *i_security;
#endif

    /* Stat data, not accessed from path walking */
    unsigned long               i_ino;
    /*
    * Filesystems may only read i_nlink directly.  They shall use the
    * following functions for modification:
    *
    *    (set|clear|inc|drop)_nlink
    *    inode_(inc|dec)_link_count
    */
    union {
        const unsigned int i_nlink;
        unsigned int __i_nlink;
    };
    dev_t                       i_rdev;
    loff_t                      i_size;
    struct timespec64   i_atime;
    struct timespec64   i_mtime;
    struct timespec64   i_ctime;
    spinlock_t          i_lock; /* i_blocks, i_bytes, maybe i_size */
    unsigned short          i_bytes;
    u8                  i_blkbits;
    u8                  i_write_hint;
    blkcnt_t            i_blocks;

#ifdef __NEED_I_SIZE_ORDERED
    seqcount_t          i_size_seqcount;
#endif

    /* Misc */
    unsigned long               i_state;
    struct rw_semaphore i_rwsem;

    unsigned long               dirtied_when;   /* jiffies of first dirtying */
    unsigned long               dirtied_time_when;

    struct hlist_node   i_hash;
```

```
    struct list_head    i_io_list;      /* backing dev IO list */
#ifdef CONFIG_CGROUP_WRITEBACK
    struct bdi_writeback        *i_wb;          /* the associated cgroup wb */

    /* foreign inode detection, see wbc_detach_inode() */
    int                 i_wb_frn_winner;
    u16                 i_wb_frn_avg_time;
    u16                 i_wb_frn_history;
#endif
    struct list_head    i_lru;          /* inode LRU list */
    struct list_head    i_sb_list;
    struct list_head    i_wb_list;      /* backing dev writeback list */
    union {
        struct hlist_head       i_dentry;
        struct rcu_head         i_rcu;
    };
    atomic64_t          i_version;
    atomic64_t          i_sequence; /* see futex */
    atomic_t            i_count;
    atomic_t            i_dio_count;
    atomic_t            i_writecount;
#if defined(CONFIG_IMA) || defined(CONFIG_FILE_LOCKING)
    atomic_t            i_readcount; /* struct files open RO */
#endif
    union {
        const struct file_operations    *i_fop; /* former ->i_op->default_file_ops */
        void (*free_inode)(struct inode *);
    };
    struct file_lock_context    *i_flctx;
    struct address_space        i_data;
    struct list_head    i_devices;
    union {
        struct pipe_inode_info  *i_pipe;
        struct cdev             *i_cdev;
        char                    *i_link;
        unsigned                i_dir_seq;
    };

    __u32                       i_generation;

#ifdef CONFIG_FSNOTIFY
    __u32                       i_fsnotify_mask; /* all events this inode cares about */
    struct fsnotify_mark_connector __rcu        *i_fsnotify_marks;
#endif

#ifdef CONFIG_FS_ENCRYPTION
    struct fscrypt_info *i_crypt_info;
#endif

#ifdef CONFIG_FS_VERITY
    struct fsverity_info        *i_verity_info;
#endif

    void                        *i_private; /* fs or device private pointer */
} __randomize_layout;
```

1.6 inode_operations

```
struct inode_operations {
    struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int);
    const char * (*get_link) (struct dentry *, struct inode *, struct delayed_call *);
    int (*permission) (struct user_namespace *, struct inode *, int);
    struct posix_acl * (*get_acl)(struct inode *, int, bool);

    int (*readlink) (struct dentry *, char __user *,int);
```

```
    int (*create) (struct user_namespace *, struct inode *,struct dentry *,
                umode_t, bool);
    int (*link) (struct dentry *,struct inode *,struct dentry *);
    int (*unlink) (struct inode *,struct dentry *);
    int (*symlink) (struct user_namespace *, struct inode *,struct dentry *,
            const char *);
    int (*mkdir) (struct user_namespace *, struct inode *,struct dentry *,
                umode_t);
    int (*rmdir) (struct inode *,struct dentry *);
    int (*mknod) (struct user_namespace *, struct inode *,struct dentry *,
                umode_t,dev_t);
    int (*rename) (struct user_namespace *, struct inode *, struct dentry *,
            struct inode *, struct dentry *, unsigned int);
    int (*setattr) (struct user_namespace *, struct dentry *,
            struct iattr *);
    int (*getattr) (struct user_namespace *, const struct path *,
            struct kstat *, u32, unsigned int);
    ssize_t (*listxattr) (struct dentry *, char *, size_t);
    int (*fiemap)(struct inode *, struct fiemap_extent_info *, u64 start,
                u64 len);
    int (*update_time)(struct inode *, struct timespec64 *, int);
    int (*atomic_open)(struct inode *, struct dentry *,
                struct file *, unsigned open_flag,
                umode_t create_mode);
    int (*tmpfile) (struct user_namespace *, struct inode *,
            struct dentry *, umode_t);
    int (*set_acl)(struct user_namespace *, struct inode *,
                struct posix_acl *, int);
    int (*fileattr_set)(struct user_namespace *mnt_userns,
                struct dentry *dentry, struct fileattr *fa);
    int (*fileattr_get)(struct dentry *dentry, struct fileattr *fa);
} ____cacheline_aligned;
```

1.7 dentry

```
struct dentry {
    /* RCU lookup touched fields */
    unsigned int d_flags;               /* protected by d_lock */
    seqcount_spinlock_t d_seq;  /* per dentry seqlock */
    struct hlist_bl_node d_hash;        /* lookup hash list */
    struct dentry *d_parent;    /* parent directory */
    struct qstr d_name;
    struct inode *d_inode;              /* Where the name belongs to - NULL is
                        * negative */
    unsigned char d_iname[DNAME_INLINE_LEN];    /* small names */

    /* Ref lookup also touches following */
    struct lockref d_lockref;   /* per-dentry lock and refcount */
    const struct dentry_operations *d_op;
    struct super_block *d_sb;   /* The root of the dentry tree */
    unsigned long d_time;               /* used by d_revalidate */
    void *d_fsdata;                     /* fs-specific data */

    union {
        struct list_head d_lru;         /* LRU list */
        wait_queue_head_t *d_wait;      /* in-lookup ones only */
    };
    struct list_head d_child;   /* child of parent list */
    struct list_head d_subdirs; /* our children */
    /*
        * d_alias and d_rcu can share memory
        */
    union {
        struct hlist_node d_alias;      /* inode alias list */
```

```
        struct hlist_bl_node d_in_lookup_hash;  /* only for in-lookup ones */
            struct rcu_head d_rcu;
    } d_u;
} __randomize_layout;
```

1.8 dentry_operations

```
struct dentry_operations {
    int (*d_revalidate)(struct dentry *, unsigned int);
    int (*d_weak_revalidate)(struct dentry *, unsigned int);
    int (*d_hash)(const struct dentry *, struct qstr *);
    int (*d_compare)(const struct dentry *,
        unsigned int, const char *, const struct qstr *);
    int (*d_delete)(const struct dentry *);
    int (*d_init)(struct dentry *);
    void (*d_release)(struct dentry *);
    void (*d_prune)(struct dentry *);
    void (*d_iput)(struct dentry *, struct inode *);
    char *(*d_dname)(struct dentry *, char *, int);
    struct vfsmount *(*d_automount)(struct path *);
    int (*d_manage)(const struct path *, bool);
    struct dentry *(*d_real)(struct dentry *, const struct inode *);
} ____cacheline_aligned;
```

1.9 file

```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        struct inode            *f_inode;       /* cached value */
        const struct file_operations    *f_op;

        /*
         * Protects f_ep, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t              f_lock;
        enum rw_hint            f_write_hint;
        atomic_long_t           f_count;
        unsigned int            f_flags;
        fmode_t                 f_mode;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;

        u64                     f_version;
#ifdef CONFIG_SECURITY
        void                    *f_security;
#endif
        /* needed for tty driver, and maybe others */
        void                    *private_data;

#ifdef CONFIG_EPOLL
        /* Used by fs/eventpoll.c to link all the hooks to this file */
        struct hlist_head       *f_ep;
#endif /* #ifdef CONFIG_EPOLL */
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err; /* for syncfs */
```

```
} __randomize_layout
  __attribute__((aligned(4)));  /* lest something weird decides that 2 is OK */
```

1.10 file_operations

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iopoll)(struct kiocb *kiocb, struct io_comp_batch *,
                        unsigned int flags);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
            unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t
            , unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t,
             unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
        unsigned (*mmap_capabilities)(struct file *);
#endif
        ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
                        loff_t, size_t, unsigned int);
        loff_t (*remap_file_range)(struct file *file_in, loff_t pos_in,
                                   struct file *file_out, loff_t pos_out,
                                   loff_t len, unsigned int remap_flags);
        int (*fadvise)(struct file *, loff_t, loff_t, int);
} __randomize_layout;
```

1.11 nameidata

```
#define EMBEDDED_LEVELS 2
struct nameidata {
        struct path     path;
        struct qstr     last;
        struct path     root;
        struct inode    *inode; /* path.dentry.d_inode */
        unsigned int    flags, state;
        unsigned        seq, m_seq, r_seq;
        int             last_type;
        unsigned        depth;
        int             total_link_count;
```

```
        struct saved {
                struct path link;
                struct delayed_call done;
                const char *name;
                unsigned seq;
        } *stack, internal[EMBEDDED_LEVELS];
        struct filename *name;
        struct nameidata *saved;
        unsigned        root_seq;
        int             dfd;
        kuid_t          dir_uid;
        umode_t         dir_mode;
} __randomize_layout;
```

1.12 path

```
struct path {
        struct vfsmount *mnt;
        struct dentry *dentry;
} __randomize_layout;
```

1.13 stat

```
struct stat {
        dev_t     st_dev;     /* ID of device containing file */
        ino_t     st_ino;     /* inode number */
        mode_t    st_mode;    /* protection */
        nlink_t   st_nlink;   /* number of hard links */
        uid_t     st_uid;     /* user ID of owner */
        gid_t     st_gid;     /* group ID of owner */
        dev_t     st_rdev;    /* device ID (if special file) */
        off_t     st_size;    /* total size, in bytes */
        blksize_t st_blksize; /* blocksize for file system I/O */
        blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
        time_t    st_atime;   /* time of last access */
        time_t    st_mtime;   /* time of last modification */
        time_t    st_ctime;   /* time of last status change */
};
```

1.14 _IO_FILE

```
struct _IO_FILE
{
    int _flags;         /* High-order word is _IO_MAGIC; rest is flags. */

    /* The following pointers correspond to the C++ streambuf protocol. */
    char *_IO_read_ptr;     /* Current read pointer */
    char *_IO_read_end;     /* End of get area. */
    char *_IO_read_base;        /* Start of putback+get area. */
    char *_IO_write_base;       /* Start of put area. */
    char *_IO_write_ptr;        /* Current put pointer. */
    char *_IO_write_end;        /* End of put area. */
    char *_IO_buf_base;     /* Start of reserve area. */
    char *_IO_buf_end;      /* End of reserve area. */

    /* The following fields are used to support backing up and undo. */
    char *_IO_save_base;    /* Pointer to start of non-current get area. */
    char *_IO_backup_base;  /* Pointer to first valid character of backup area */
    char *_IO_save_end;     /* Pointer to end of non-current get area. */

    struct _IO_marker *_markers;
    struct _IO_FILE *_chain;
```

```
    int _fileno;
    int _flags2;
    __off_t _old_offset; /* This used to be _offset but it's too small.  */

    /* 1+column number of pbase(); 0 is unknown. */
    unsigned short _cur_column;
    signed char _vtable_offset;
    char _shortbuf[1];

    _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};

struct _IO_FILE_complete
{
    struct _IO_FILE _file;
#endif
    __off64_t _offset;
    /* Wide character stream stuff.  */
    struct _IO_codecvt *_codecvt;
    struct _IO_wide_data *_wide_data;
    struct _IO_FILE *_freeres_list;
    void *_freeres_buf;
    size_t __pad5;
    int _mode;
    /* Make sure we don't get into trouble again.  */
    char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
};
```

2 Proc VFS

2.1 proc_dir_entry

```
/*
 * This is not completely implemented yet. The idea is to
 * create an in-memory tree (like the actual /proc filesystem
 * tree) of these proc_dir_entries, so that we can dynamically
 * add new files to /proc.
 *
 * parent/subdir are used for the directory structure (every /proc file has a
 * parent, but "subdir" is empty for all non-directory entries).
 * subdir_node is used to build the rb tree "subdir" of the parent.
 */
struct proc_dir_entry {
        /*
         * number of callers into module in progress;
         * negative -> it's going away RSN
         */
        atomic_t in_use;
        refcount_t refcnt;
        struct list_head pde_openers;   /* who did ->open, but not ->release */
        /* protects ->pde_openers and all struct pde_opener instances */
        spinlock_t pde_unload_lock;
        struct completion *pde_unload_completion;
        const struct inode_operations *proc_iops;
        union {
                const struct proc_ops *proc_ops;
                const struct file_operations *proc_dir_ops;
        };
        const struct dentry_operations *proc_dops;
        union {
                const struct seq_operations *seq_ops;
                int (*single_show)(struct seq_file *, void *);
        };
```

```
        proc_write_t write;
        void *data;
        unsigned int state_size;
        unsigned int low_ino;
        nlink_t nlink;
        kuid_t uid;
        kgid_t gid;
        loff_t size;
        struct proc_dir_entry *parent;
        struct rb_root subdir;
        struct rb_node subdir_node;
        char *name;
        umode_t mode;
        u8 flags;
        u8 namelen;
        char inline_name[];
} __randomize_layout;
```

2.2 proc_ops

```
struct proc_ops {
        unsigned int proc_flags;
        int     (*proc_open)(struct inode *, struct file *);
        ssize_t (*proc_read)(struct file *, char __user *, size_t, loff_t *);
        ssize_t (*proc_read_iter)(struct kiocb *, struct iov_iter *);
        ssize_t (*proc_write)(struct file *, const char __user *, size_t, loff_t *);
        /* mandatory unless nonseekable_open() or equivalent is used */
        loff_t  (*proc_lseek)(struct file *, loff_t, int);
        int     (*proc_release)(struct inode *, struct file *);
        __poll_t (*proc_poll)(struct file *, struct poll_table_struct *);
        long    (*proc_ioctl)(struct file *, unsigned int, unsigned long);
#ifdef CONFIG_COMPAT
        long    (*proc_compat_ioctl)(struct file *, unsigned int, unsigned long);
#endif
        int     (*proc_mmap)(struct file *, struct vm_area_struct *);
        unsigned long (*proc_get_unmapped_area)(struct file *, unsigned long, unsigned
            long, unsigned long, unsigned long);
} __randomize_layout;
```

2.3 seq_file

```
struct seq_file {
        char *buf;
        size_t size;
        size_t from;
        size_t count;
        size_t pad_until;
        loff_t index;
        loff_t read_pos;
        struct mutex lock;
        const struct seq_operations *op;
        int poll_event;
        const struct file *file;
        void *private;
};
```

2.4 seq_operations

```
struct seq_operations {
        void * (*start) (struct seq_file *m, loff_t *pos);
        void (*stop) (struct seq_file *m, void *v);
        void * (*next) (struct seq_file *m, void *v, loff_t *pos);
        int (*show) (struct seq_file *m, void *v);
```

```
};
```

3 Sockets

3.1 socket

```
enum sock_type {
    SOCK_STREAM = 1,
    SOCK_DGRAM  = 2,
    SOCK_RAW    = 3,
    SOCK_RDM    = 4,
    SOCK_SEQPACKET      = 5,
    SOCK_DCCP   = 6,
    SOCK_PACKET = 10,
};

struct socket {
        socket_state                state;
        short                           type;
        unsigned long               flags;
        struct file                     *file;
        struct sock                     *sk;
        const struct proto_ops  *ops;
        struct socket_wq            wq;
};
```

3.2 sockaddr

```
/* Supported address families. */
#define AF_UNSPEC       0
#define AF_UNIX         1       /* Unix domain sockets          */
#define AF_LOCAL        1       /* POSIX name for AF_UNIX       */
#define AF_INET         2       /* Internet IP Protocol         */
/* ... */
#define AF_INET6        10      /* IP version 6                 */

struct sockaddr {
        sa_family_t     sa_family;          /* address family, AF_xxx   */
        char            sa_data[14];    /* 14 bytes of protocol address */
};
```

3.3 sockaddr_un

```
#define UNIX_PATH_MAX   108

struct sockaddr_un {
        __kernel_sa_family_t sun_family; /* AF_UNIX */
        char sun_path[UNIX_PATH_MAX];   /* pathname */
};
```

3.4 sockaddr_in

```
struct sockaddr_in {
  __kernel_sa_family_t  sin_family;     /* Address family               */
  __be16                sin_port;       /* Port number                  */
  struct in_addr        sin_addr;       /* Internet address             */

  /* Pad to size of `struct sockaddr'. */
  unsigned char         __pad[__SOCK_SIZE__ - sizeof(short int) -
                        sizeof(unsigned short int) - sizeof(struct in_addr)];
};
```

```
/* Internet address. */
struct in_addr {
        __be32  s_addr; /* unsigned int */
};
```

4 Interrupts

4.1 softirq_action

```
struct softirq_action
{
        void    (*action)(struct softirq_action *);
};
```

4.2 tasklet_struct

```
struct tasklet_struct
{
        struct tasklet_struct *next;
        unsigned long state;
        atomic_t count;
        bool use_callback;
        union {
                void (*func)(unsigned long data);
                void (*callback)(struct tasklet_struct *t);
        };
        unsigned long data;
};

#define DECLARE_TASKLET(name, _callback)                \
    struct tasklet_struct name = {                              \
            .count = ATOMIC_INIT(0),                    \
            .callback = _callback,                              \
            .use_callback = true,                               \
        }
```

4.3 work_struct

```
struct work_struct {
        atomic_long_t data;
        struct list_head entry;
        work_func_t func;
#ifdef CONFIG_LOCKDEP
        struct lockdep_map lockdep_map;
#endif
};
```

4.4 worker

```
/*
 * The poor guys doing the actual heavy lifting.  All on-duty workers are
 * either serving the manager role, on idle list or on busy hash.  For
 * details on the locking annotation (L, I, X...), refer to workqueue.c.
 *
 * Only to be used in workqueue and async.
 */
struct worker {
        /* on idle list while idle, on busy hash table while busy */
        union {
                struct list_head        entry;  /* L: while idle */
```

```
                struct hlist_node       hentry; /* L: while busy */
        };

        struct work_struct      *current_work;  /* L: work being processed */
        work_func_t             current_func;   /* L: current_work's fn */
        struct pool_workqueue   *current_pwq;   /* L: current_work's pwq */
        unsigned int            current_color;  /* L: current_work's color */
        struct list_head        scheduled;      /* L: scheduled works */

        /* 64 bytes boundary on 64bit, 32 on 32bit */

        struct task_struct      *task;          /* I: worker task */
        struct worker_pool      *pool;          /* A: the associated pool */
                                                /* L: for rescuers */
        struct list_head        node;           /* A: anchored at pool->workers */
                                                /* A: runs through worker->node */

        unsigned long           last_active;    /* L: last active timestamp */
        unsigned int            flags;          /* X: flags */
        int                     id;             /* I: worker id */
        int                     sleeping;       /* None */

        /*
         * Opaque string set with work_set_desc().  Printed out with task
         * dump for debugging - WARN, BUG, panic or sysrq.
         */
        char                    desc[WORKER_DESC_LEN];

        /* used only by rescuers to point to the target workqueue */
        struct workqueue_struct *rescue_wq;     /* I: the workqueue to rescue */

        /* used by the scheduler to determine a worker's last known identity */
        work_func_t             last_func;
};
```

4.5 workqueue_struct

```
/*
 * The externally visible workqueue.  It relays the issued work items to
 * the appropriate worker_pool through its pool_workqueues.
 */
struct workqueue_struct {
        struct list_head        pwqs;           /* WR: all pwqs of this wq */
        struct list_head        list;           /* PR: list of all workqueues */

        struct mutex            mutex;          /* protects this wq */
        int                     work_color;     /* WQ: current work color */
        int                     flush_color;    /* WQ: current flush color */
        atomic_t                nr_pwqs_to_flush; /* flush in progress */
        struct wq_flusher       *first_flusher; /* WQ: first flusher */
        struct list_head        flusher_queue;  /* WQ: flush waiters */
        struct list_head        flusher_overflow; /* WQ: flush overflow list */

        struct list_head        maydays;        /* MD: pwqs requesting rescue */
        struct worker           *rescuer;       /* MD: rescue worker */

        int                     nr_drainers;    /* WQ: drain in progress */
        int                     saved_max_active; /* WQ: saved pwq max_active */

        struct workqueue_attrs  *unbound_attrs; /* PW: only for unbound wqs */
        struct pool_workqueue   *dfl_pwq;       /* PW: only for unbound wqs */

#ifdef CONFIG_SYSFS
        struct wq_device        *wq_dev;        /* I: for sysfs interface */
#endif
```

```
#ifdef CONFIG_LOCKDEP
        char                    *lock_name;
        struct lock_class_key   key;
        struct lockdep_map      lockdep_map;
#endif
        char                    name[WQ_NAME_LEN]; /* I: workqueue name */

        /*
         * Destruction of workqueue_struct is RCU protected to allow walking
         * the workqueues list without grabbing wq_pool_mutex.
         * This is used to dump all workqueues from sysrq.
         */
        struct rcu_head         rcu;

        /* hot fields used during command issue, aligned to cacheline */
        unsigned int            flags ____cacheline_aligned; /* WQ: WQ_* flags */
        struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs */
        struct pool_workqueue __rcu *numa_pwq_tbl[]; /* PWR: unbound pwqs indexed by node
             */
};
```

4.6 pool_workqueue

```
/*
 * The per-pool workqueue.  While queued, the lower WORK_STRUCT_FLAG_BITS
 * of work_struct->data are used for flags and the remaining high bits
 * point to the pwq; thus, pwqs need to be aligned at two's power of the
 * number of flag bits.
 */
struct pool_workqueue {
        struct worker_pool      *pool;          /* I: the associated pool */
        struct workqueue_struct *wq;            /* I: the owning workqueue */
        int                     work_color;     /* L: current color */
        int                     flush_color;    /* L: flushing color */
        int                     refcnt;         /* L: reference count */
        int                     nr_in_flight[WORK_NR_COLORS];
                                                /* L: nr of in_flight works */

        int                     nr_active;      /* L: nr of active works */
        int                     max_active;     /* L: max active works */
        struct list_head        inactive_works; /* L: inactive works */
        struct list_head        pwqs_node;      /* WR: node on wq->pwqs */
        struct list_head        mayday_node;    /* MD: node on wq->maydays */

        struct work_struct      unbound_release_work;
        struct rcu_head         rcu;
} __aligned(1 << WORK_STRUCT_FLAG_BITS);
```

4.7 worker_pool

```
struct worker_pool {
        raw_spinlock_t          lock;           /* the pool lock */
        int                     cpu;            /* I: the associated cpu */
        int                     node;           /* I: the associated node ID */
        int                     id;             /* I: pool ID */
        unsigned int            flags;          /* X: flags */

        unsigned long           watchdog_ts;    /* L: watchdog timestamp */

        /* The current concurrency level. */
        atomic_t                nr_running;

        struct list_head        worklist;       /* L: list of pending works */
```

```
        int                     nr_workers;     /* L: total number of workers */
        int                     nr_idle;        /* L: currently idle workers */

        struct list_head        idle_list;      /* X: list of idle workers */
        struct timer_list       idle_timer;     /* L: worker idle timeout */
        struct timer_list       mayday_timer;   /* L: SOS timer for workers */

        /* a workers is either on busy_hash or idle_list, or the manager */
        DECLARE_HASHTABLE(busy_hash, BUSY_WORKER_HASH_ORDER);
                                                /* L: hash of busy workers */

        struct worker           *manager;       /* L: purely informational */
        struct list_head        workers;        /* A: attached workers */
        struct completion       *detach_completion; /* all workers detached */

        struct ida              worker_ida;     /* worker IDs for task name */

        struct workqueue_attrs  *attrs;         /* I: worker attributes */
        struct hlist_node       hash_node;      /* PL: unbound_pool_hash node */
        int                     refcnt;         /* PL: refcnt for unbound pools */

        /*
         * Destruction of pool is RCU protected to allow dereferences
         * from get_work_pool().
         */
        struct rcu_head         rcu;
};
```

5 Devices

5.1 device

```
struct device {
        struct kobject kobj;
        struct device           *parent;

        struct device_private   *p;

        const char              *init_name; /* initial name of the device */
        const struct device_type *type;

        struct bus_type *bus;           /* type of bus device is on */
        struct device_driver *driver;   /* which driver has allocated this
                                           device */
        void            *platform_data; /* Platform specific data, device
                                           core doesn't touch it */
        void            *driver_data;   /* Driver data, set and get with
                                           dev_set_drvdata/dev_get_drvdata */
#ifdef CONFIG_PROVE_LOCKING
        struct mutex            lockdep_mutex;
#endif
        struct mutex            mutex;  /* mutex to synchronize calls to
                                         * its driver.
                                         */

        struct dev_links_info   links;
        struct dev_pm_info      power;
        struct dev_pm_domain    *pm_domain;

#ifdef CONFIG_ENERGY_MODEL
        struct em_perf_domain   *em_pd;
#endif

#ifdef CONFIG_PINCTRL
```

```
        struct dev_pin_info     *pins;
#endif
        struct dev_msi_info     msi;
#ifdef CONFIG_DMA_OPS
        const struct dma_map_ops *dma_ops;
#endif
        u64             *dma_mask;      /* dma mask (if dma'able device) */
        u64             coherent_dma_mask;/* Like dma_mask, but for
                                             alloc_coherent mappings as
                                             not all hardware supports
                                             64 bit addresses for consistent
                                             allocations such descriptors. */
        u64             bus_dma_limit;  /* upstream dma constraint */
        const struct bus_dma_region *dma_range_map;

        struct device_dma_parameters *dma_parms;

        struct list_head        dma_pools;      /* dma pools (if dma'ble) */

#ifdef CONFIG_DMA_DECLARE_COHERENT
        struct dma_coherent_mem *dma_mem; /* internal for coherent mem
                                             override */
#endif
#ifdef CONFIG_DMA_CMA
        struct cma *cma_area;           /* contiguous memory area for dma
                                           allocations */
#endif
#ifdef CONFIG_SWIOTLB
        struct io_tlb_mem *dma_io_tlb_mem;
#endif
        /* arch specific additions */
        struct dev_archdata     archdata;

        struct device_node      *of_node; /* associated device tree node */
        struct fwnode_handle    *fwnode; /* firmware device node */

#ifdef CONFIG_NUMA
        int             numa_node;      /* NUMA node this device is close to */
#endif
        dev_t                   devt;   /* dev_t, creates the sysfs "dev" */
        u32                     id;     /* device instance */

        spinlock_t              devres_lock;
        struct list_head        devres_head;

        struct class            *class;
        const struct attribute_group **groups;  /* optional groups */

        void    (*release)(struct device *dev);
        struct iommu_group      *iommu_group;
        struct dev_iommu        *iommu;

        enum device_removable   removable;

        bool                    offline_disabled:1;
        bool                    offline:1;
        bool                    of_node_reused:1;
        bool                    state_synced:1;
        bool                    can_match:1;
#if defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_DEVICE) || \
    defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU) || \
    defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU_ALL)
        bool                    dma_coherent:1;
#endif
#ifdef CONFIG_DMA_OPS_BYPASS
        bool                    dma_ops_bypass : 1;
```

```
#endif
};
```

5.2 device_driver

```
struct device_driver {
        const char              *name;
        struct bus_type         *bus;

        struct module           *owner;
        const char              *mod_name;      /* used for built-in modules */

        bool suppress_bind_attrs;       /* disables bind/unbind via sysfs */
        enum probe_type probe_type;

        const struct of_device_id       *of_match_table;
        const struct acpi_device_id     *acpi_match_table;

        int (*probe) (struct device *dev);
        void (*sync_state)(struct device *dev);
        int (*remove) (struct device *dev);
        void (*shutdown) (struct device *dev);
        int (*suspend) (struct device *dev, pm_message_t state);
        int (*resume) (struct device *dev);
        const struct attribute_group **groups;
        const struct attribute_group **dev_groups;

        const struct dev_pm_ops *pm;
        void (*coredump) (struct device *dev);

        struct driver_private *p;
};
```

6 Sheduling

6.1 task_struct

```
struct task_struct {
        struct thread_info              thread_info;
        unsigned int                    __state;

        void                            *stack;
        refcount_t                      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int                    flags;
        unsigned int                    ptrace;

        /* smp fields skipped... */

        int                             on_rq; /* run queue (per CPU) */

        int                             prio;
        int                             static_prio;
        int                             normal_prio;
        unsigned int                    rt_priority;

        /* ... */

        struct sched_info               sched_info;

        struct list_head                tasks;

        /* ... */
```

```
        struct mm_struct                *mm; /* memory mapping */
        struct mm_struct                *active_mm;

        int                             exit_state;
        int                             exit_code;
        int                             exit_signal;

        /* The signal sent when the parent dies: */
        int                             pdeath_signal;

        /* ... */

        pid_t                           pid;
        pid_t                           tgid;

        /* Real parent process: */
        struct task_struct __rcu        *real_parent;

        /* Recipient of SIGCHLD, wait4() reports: */
        struct task_struct __rcu        *parent;

        struct list_head                children;
        struct list_head                sibling;
        struct task_struct              *group_leader;

        /* PID/PID hash table linkage. */
        struct pid                      *thread_pid;
        struct hlist_node               pid_links[PIDTYPE_MAX];
        struct list_head                thread_group;
        struct list_head                thread_node;

        /* PF_KTHREAD | PF_IO_WORKER */
        void                            *worker_private;

        /*
         * executable name, excluding path.
         *
         * - normally initialized setup_new_exec()
         * - access it with [gs]et_task_comm()
         * - lock it with task_lock()
         */
        char                            comm[TASK_COMM_LEN];

        struct nameidata                *nameidata;

#ifdef CONFIG_SYSVIPC /* System V IPC related stuff */
        struct sysv_sem                 sysvsem;
        struct sysv_shm                 sysvshm;
#endif

        /* Filesystem information: */
        struct fs_struct                *fs;

        /* Open file information: */
        struct files_struct             *files;

        /* Namespaces: */
        struct nsproxy                  *nsproxy;

        /* Signal handlers: */
        struct signal_struct            *signal;
        struct sighand_struct __rcu             *sighand;
        sigset_t                        blocked;
        sigset_t                        real_blocked;
        /* Restored if set_restore_sigmask() was used: */
```

```
        sigset_t                        saved_sigmask;
        struct sigpending               pending;

#ifdef CONFIG_AUDIT
#ifdef CONFIG_AUDITSYSCALL
        struct audit_context            *audit_context;
#endif
        kuid_t                          loginuid;
        unsigned int                    sessionid;
#endif

        /* Protection against (de-)allocation: mm, files, fs, tty, keyrings, mems_allowed
            , mempolicy: */
        spinlock_t                      alloc_lock;

        /* ... */

        /*
         * I/O subsystem state of the associated processes.  It is refcounted
         * and kmalloc'ed. These could be shared between processes.
         */
        struct io_context               *io_context;

        /* ... */

        /* CPU-specific state of this task: */
        struct thread_struct            thread;
};
```

6.2 fs_struct

```
struct fs_struct {
        int users;
        spinlock_t lock;
        seqcount_spinlock_t seq;
        int umask;
        int in_exec;
        struct path root, pwd;
} __randomize_layout;
```

6.3 fdtable

```
struct fdtable {
        unsigned int max_fds;
        struct file __rcu **fd;      /* current fd array */
        unsigned long *close_on_exec;
        unsigned long *open_fds;
        unsigned long *full_fds_bits;
        struct rcu_head rcu;
};
```

6.4 nsproxy

```
/*
 * A structure to contain pointers to all per-process
 * namespaces - fs (mount), uts, network, sysvipc, etc.
 *
 * The pid namespace is an exception -- it's accessed using
 * task_active_pid_ns.  The pid namespace here is the
 * namespace that children will use.
 *
 * 'count' is the number of tasks holding a reference.
 * The count for each namespace, then, will be the number
```

```
 * of nsproxies pointing to it, not the number of tasks.
 *
 * The nsproxy is shared by tasks which share all namespaces.
 * As soon as a single namespace is cloned or unshared, the
 * nsproxy is copied.
 */
struct nsproxy {
        atomic_t count;
        struct uts_namespace *uts_ns;
        struct ipc_namespace *ipc_ns;
        struct mnt_namespace *mnt_ns;
        struct pid_namespace *pid_ns_for_children;
        struct net           *net_ns;
        struct time_namespace *time_ns;
        struct time_namespace *time_ns_for_children;
        struct cgroup_namespace *cgroup_ns;
};
extern struct nsproxy init_nsproxy;
```