```
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_archdataarchdata;

        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
};
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;
};

#define MAJOR(dev)      ((unsigned int) ((dev) >> MINORBITS))
#define MINOR(dev)      ((unsigned int) ((dev) & MINORMASK))
#define MKDEV(ma,mi)    (((ma) << MINORBITS) | (mi))

* @from: the first in the desired range of device numbers; must include the major number.
 * @count: the number of consecutive device numbers required
 * @name: the name of the device or driver.
* Return value is zero on success, a negative error code on failure.
int register_chrdev_region(dev_t from, unsigned count, const char *name)


* @dev: output parameter for first assigned number
 * @baseminor: first of the requested range of minor numbers
 * @count: the number of minor numbers required
 * @name: the name of the associated device or driver
int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count,
                        const char *name)

* @from: the first in the range of numbers to unregister
 * @count: the number of device numbers to unregister
 *
 * This function will unregister a range of @count device numbers,
 * starting with @from.  The caller should normally be the one who
 * allocated those numbers in the first place...
void unregister_chrdev_region(dev_t from, unsigned count)

request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
            const char *name, void *dev)
const void *free_irq(unsigned int irq, void *dev_id)

# IRQF_SHARED - allow sharing the irq among several devices
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,                       \
}

enum
{
        TASKLET_STATE_SCHED,        /* Tasklet is scheduled for execution */
        TASKLET_STATE_RUN       /* Tasklet is running (SMP only) */
};
static inline void tasklet_schedule(struct tasklet_struct *t);
static inline void tasklet_hi_schedule(struct tasklet_struct *t);
static inline void tasklet_disable(struct tasklet_struct *t);
static inline void tasklet_enable(struct tasklet_struct *t);
void tasklet_init(struct tasklet_struct *t, void (*func)(unsigned long), unsigned long data);
void tasklet_kill(struct tasklet_struct *t);

struct block_device {
        sector_t            bd_start_sect;
        sector_t            bd_nr_sectors;
        struct disk_stats __percpu *bd_stats;
        unsigned long               bd_stamp;
        bool                        bd_read_only;   /* read-only policy */
        dev_t                       bd_dev;
        int                         bd_openers;
        struct inode *              bd_inode;       /* will die */
        struct super_block *        bd_super;
        void *                      bd_claiming;
        struct device               bd_device;
        void *                      bd_holder;
        int                         bd_holders;
        bool                        bd_write_holder;
        struct kobject              *bd_holder_dir;
        u8                          bd_partno;
        spinlock_t                  bd_size_lock; /* for bd_inode->i_size updates */
        struct gendisk *        bd_disk;
        struct request_queue *      bd_queue;

        /* The counter of freeze processes */
        int                         bd_fsfreeze_count;
        /* Mutex for freeze */
        struct mutex                bd_fsfreeze_mutex;
        struct super_block  *bd_fsfreeze_sb;

        struct partition_meta_info *bd_meta_info;
#ifdef CONFIG_FAIL_MAKE_REQUEST
        bool                        bd_make_it_fail;
#endif
} __randomize_layout;

struct cdev {
        struct kobject kobj;
        struct module *owner;
        const struct file_operations *ops;
        struct list_head list;
```

```
        dev_t dev;
        unsigned int count;
} __randomize_layout;

struct inode {
        umode_t                 i_mode; // mode
        unsigned short          i_opflags;
        kuid_t                  i_uid; // uid
        kgid_t                  i_gid; // 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; // 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;

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

#define DECLARE_WORK(name, void (*f)(void *))
#define INIT_WORK(struct work_struct *work, void (*f)(void), void *)
static inline bool queue_work(struct workqueue_struct *wq, struct work_struct *work);
```

```
void flush_workqueue(struct workqueue_struct *wq);
extern void destroy_workqueue(struct workqueue_struct *wq);
void open_softirq(int nr, void (*action)(struct softirq_action *))
{
        softirq_vec[nr].action = action;
}
void raise_softirq(unsigned int nr)
{
        unsigned long flags;

        local_irq_save(flags);
        raise_softirq_irqoff(nr);
        local_irq_restore(flags);
}
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;
struct proc_dir_entry *proc_create(const char *name, umode_t mode, struct proc_dir_entry *parent,
const struct proc_ops *proc_ops);
extern struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *);
extern struct proc_dir_entry *proc_symlink(const char *,
                struct proc_dir_entry *, const char *);
```

```
extern void remove_proc_entry(const char *, struct proc_dir_entry *);
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;
unsigned long copy_to_user(void __user *to, const void *from, unsigned long n);

struct file_system_type {
        const char *name; /*= 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 *); /* = mount*/
        void (*kill_sb) (struct super_block *); /* =  killsb */
        struct module *owner; /* = this module */
        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;
};

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; /* = page_shift */
        unsigned long           s_blocksize; /* = page_size */
        loff_t                  s_maxbytes;             /* Max file size */
        struct file_system_type *s_type; // type
        const struct super_operations *s_op; /* = ops */
        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; /*= magic*/
        struct dentry           *s_root; /*= 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;

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;

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

        /*
         * Protects f_ep, f_flags.
         * Must not be taken from IRQ context.
         */
        spinlock_t                      f_lock;
        atomic_long_t                   f_count;
        unsigned int                    f_flags;
        fmode_t                         f_mode; // mode
        struct mutex                    f_pos_lock;
        loff_t                          f_pos; // 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 */
```

```
struct file_handle {
        __u32 handle_bytes;
        int handle_type;
        /* file identifier */
        unsigned char f_handle[];
};
extern struct dentry *mount_bdev(struct file_system_type *fs_type,
        int flags, const char *dev_name, void *data,
        int (*fill_super)(struct super_block *, void *, int));
extern struct dentry *mount_nodev(struct file_system_type *fs_type,
        int flags, void *data,
        int (*fill_super)(struct super_block *, void *, int));
struct kmem_cache *kmem_cache_create(const char *name, unsigned int size,
                                     unsigned int align, slab_flags_t flags,
                                     void (*ctor)(void *));
void kmem_cache_destroy(struct kmem_cache *s);
void *kmem_cache_alloc(struct kmem_cache *s, gfp_t flags) // GFP_KERNEL
void kmem_cache_free(struct kmem_cache *s, void *objp);
extern int register_filesystem(struct file_system_type *);
extern int unregister_filesystem(struct file_system_type *);
extern struct inode *new_inode(struct super_block *sb);
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 *); = generic_delete_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 *); = simple_stat_fs
        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 *);
};
int generic_delete_inode(struct inode *inode)
{
        return 1;
```

```
}
int simple_statfs(struct dentry *dentry, struct kstatfs *buf)
{
        buf->f_type = dentry->d_sb->s_magic;
        buf->f_bsize = PAGE_SIZE;
        buf->f_namelen = NAME_MAX;
        return 0;
}
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;
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;

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;
};
typedef enum {
        SS_FREE = 0,                    /* not allocated                */
        SS_UNCONNECTED,                         /* unconnected to any socket    */
        SS_CONNECTING,                          /* in process of connecting     */
        SS_CONNECTED,                   /* connected to socket                  */
        SS_DISCONNECTING                /* in process of disconnecting  */
} socket_state;

struct sockaddr {
        sa_family_t             sa_family;      /* address family, AF_xxx       */
        char                    sa_data[14];    /* 14 bytes of protocol address */
};
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)];
};
Sys_socket -> sock_create
#define UNIX_PATH_MAX  108
struct sockaddr_un {
        __kernel_sa_family_t sun_family; /* AF_UNIX */
        char sun_path[UNIX_PATH_MAX];   /* pathname */
};
#define ___htonl(x) __cpu_to_be32(x)
#define ___htons(x) __cpu_to_be16(x)
#define ___ntohl(x) __be32_to_cpu(x)
#define ___ntohs(x) __be16_to_cpu(x)

#define htonl(x) ___htonl(x)
#define ntohl(x) ___ntohl(x)
#define htons(x) ___htons(x)
#define ntohs(x) ___ntohs(x)
```

```
struct proto_ops {
        int             family;
        struct module   *owner;
        int             (*release)   (struct socket *sock);
        int             (*bind)      (struct socket *sock,
                                      struct sockaddr *myaddr,
                                      int sockaddr_len);
        int             (*connect)   (struct socket *sock,
                                      struct sockaddr *vaddr,
                                      int sockaddr_len, int flags);
        int             (*socketpair)(struct socket *sock1,
                                      struct socket *sock2);
        int             (*accept)    (struct socket *sock,
                                      struct socket *newsock, int flags, bool kern);
        int             (*getname)   (struct socket *sock,
                                      struct sockaddr *addr,
                                      int peer);
        __poll_t  (*poll)        (struct file *file, struct socket *sock,
                                      struct poll_table_struct *wait);
        int             (*ioctl)     (struct socket *sock, unsigned int cmd,
                                      unsigned long arg);
#ifdef CONFIG_COMPAT
        int             (*compat_ioctl) (struct socket *sock, unsigned int cmd,
                                      unsigned long arg);
#endif
        int             (*gettstamp) (struct socket *sock, void __user *userstamp,
                                      bool timeval, bool time32);
        int             (*listen)    (struct socket *sock, int len);
        int             (*shutdown)  (struct socket *sock, int flags);
        int             (*setsockopt)(struct socket *sock, int level,
                                      int optname, sockptr_t optval,
                                      unsigned int optlen);
        int             (*getsockopt)(struct socket *sock, int level,
                                      int optname, char __user *optval, int __user *optlen);
        void            (*show_fdinfo)(struct seq_file *m, struct socket *sock);
        int             (*sendmsg)   (struct socket *sock, struct msghdr *m,
                                      size_t total_len);
        /* Notes for implementing recvmsg:
         * ===============================
         * msg->msg_namelen should get updated by the recvmsg handlers
         * iff msg_name != NULL. It is by default 0 to prevent
         * returning uninitialized memory to user space.  The recvfrom
         * handlers can assume that msg.msg_name is either NULL or has
         * a minimum size of sizeof(struct sockaddr_storage).
         */
        int             (*recvmsg)   (struct socket *sock, struct msghdr *m,
                                      size_t total_len, int flags);
        int             (*mmap)      (struct file *file, struct socket *sock,
                                      struct vm_area_struct * vma);
        ssize_t         (*sendpage)  (struct socket *sock, struct page *page,
                                      int offset, size_t size, int flags);
        ssize_t     (*splice_read)(struct socket *sock,  loff_t *ppos,
                                       struct pipe_inode_info *pipe, size_t len, unsigned int
flags);
        int             (*set_peek_off)(struct sock *sk, int val);
        int             (*peek_len)(struct socket *sock);
```

```
        /* The following functions are called internally by kernel with
         * sock lock already held.
         */
        int             (*read_sock)(struct sock *sk, read_descriptor_t *desc,
                                     sk_read_actor_t recv_actor);
        int             (*sendpage_locked)(struct sock *sk, struct page *page,
                                           int offset, size_t size, int flags);
        int             (*sendmsg_locked)(struct sock *sk, struct msghdr *msg,
                                          size_t size);
        int             (*set_rcvlowat)(struct sock *sk, int val);
};

extern struct task_struct init_task;
#define next_task(p) \
        list_entry_rcu((p)->tasks.next, struct task_struct, tasks)

struct task_struct {
        * describes a process running in the system, created dynamically. */
        int                             prio;
        int                             static_prio;
        char comm[TASK_COMM_LEN]; /* executable name excluding path
        struct list_head                tasks;
        ...
        struct mm_struct                *mm;
        struct mm_struct                *active_mm;
        ...
        pid_t                           pid;
        pid_t                           tgid;
        ...
        /*
         * Children/sibling form the list of natural children:
         */
        struct list_head                children;
        struct list_head                sibling;
        ...
        /* Filesystem information: */
        struct fs_struct                *fs;

        /* Open file information: */
        struct files_struct             *files;
        /* Namespaces: */
        struct nsproxy                  *nsproxy;
};



struct fs_struct {
        /* information about the file system to which the process belongs*/
        int users;
        spinlock_t lock;
        seqcount_spinlock_t seq;
        int umask;
        int in_exec;
        struct path root, pwd; the mounting object of the root directory and working directory
} __randomize_layout;
/*
 * Open file table structure
```

```
 */
struct files_struct {
  /*
   * read mostly part
   */
        atomic_t count;
        bool resize_in_progress;
        wait_queue_head_t resize_wait;

        struct fdtable __rcu *fdt;
        struct fdtable fdtab;
  /*
   * written part on a separate cache line in SMP
   */
        spinlock_t file_lock ____cacheline_aligned_in_smp;
        unsigned int next_fd;
        unsigned long close_on_exec_init[1]; // fd that should be closed when exec() is called
        unsigned long open_fds_init[1]; // initial set of fd
        unsigned long full_fds_bits_init[1];
        struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```