Linux kernel v.5.15.32

```
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
#ifdef 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 */
	/*
	 * Saved pool identifier for cleancache (-1 means none)
	 */
	int cleancache_poolid;
	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 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 *);
};


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_THP_SUPPORT          8192    /* Remove once all fs converted */
#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;
};

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;
```

```
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 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 dentry_stat_t {
        long nr_dentry;
        long nr_unused;
        long age_limit;         /* age in seconds */
        long want_pages;/* pages requested by system */
        long nr_negative;       /* # of unused negative dentries */
        long dummy;             /* Reserved for future use */
};
extern struct dentry_stat_t dentry_stat;

/*
 * 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_headi_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_headi_lru;          /* inode LRU list */
        struct list_headi_sb_list;
        struct list_headi_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_headi_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 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;

/*
 * Inode flags - they have no relation to superblock flags now
 */
#define S_SYNC          (1 << 0)  /* Writes are synced at once */
#define S_NOATIME       (1 << 1)  /* Do not update access times */
#define S_APPEND(1 << 2)  /* Append-only file */
#define S_IMMUTABLE     (1 << 3)  /* Immutable file */
#define S_DEAD          (1 << 4)  /* removed, but still open directory */
#define S_NOQUOTA       (1 << 5)  /* Inode is not counted to quota */
#define S_DIRSYNC       (1 << 6)  /* Directory modifications are synchronous */
#define S_NOCMTIME      (1 << 7)  /* Do not update file c/mtime */
#define S_SWAPFILE      (1 << 8)  /* Do not truncate: swapon got its bmaps */
#define S_PRIVATE       (1 << 9)  /* Inode is fs-internal */
#define S_IMA           (1 << 10) /* Inode has an associated IMA struct */
#define S_AUTOMOUNT     (1 << 11) /* Automount/referral quasi-directory */
#define S_NOSEC         (1 << 12) /* no suid or xattr security attributes */
#ifdef CONFIG_FS_DAX
#define S_DAX           (1 << 13) /* Direct Access, avoiding the page cache */
#else
#define S_DAX           0         /* Make all the DAX code disappear */
#endif
#define S_ENCRYPTED     (1 << 14) /* Encrypted file (using fs/crypto/) */
#define S_CASEFOLD      (1 << 15) /* Casefolded file */
#define S_VERITY(1 << 16) /* Verity file (using fs/verity/) */


/**
 * struct address_space - Contents of a cacheable, mappable object.
 * @host: Owner, either the inode or the block_device.
 * @i_pages: Cached pages.
 * @invalidate_lock: Guards coherency between page cache contents and
 *   file offset->disk block mappings in the filesystem during invalidates.
 *   It is also used to block modification of page cache contents through
 *   memory mappings.
 * @gfp_mask: Memory allocation flags to use for allocating pages.
 * @i_mmap_writable: Number of VM_SHARED mappings.
 * @nr_thps: Number of THPs in the pagecache (non-shmem only).
 * @i_mmap: Tree of private and shared mappings.
 * @i_mmap_rwsem: Protects @i_mmap and @i_mmap_writable.
 * @nrpages: Number of page entries, protected by the i_pages lock.
 * @writeback_index: Writeback starts here.
 * @a_ops: Methods.
 * @flags: Error bits and flags (AS_*).
 * @wb_err: The most recent error which has occurred.
 * @private_lock: For use by the owner of the address_space.
 * @private_list: For use by the owner of the address_space.
 * @private_data: For use by the owner of the address_space.
 */
struct address_space {
        struct inode            *host;
        struct xarray           i_pages;
        struct rw_semaphore     invalidate_lock;
        gfp_t                   gfp_mask;
        atomic_t        i_mmap_writable;
#ifdef CONFIG_READ_ONLY_THP_FOR_FS
        /* number of thp, only for non-shmem files */
        atomic_t        nr_thps;
#endif
        struct rb_root_cached   i_mmap;
        struct rw_semaphore     i_mmap_rwsem;
        unsigned long           nrpages;
        pgoff_t                 writeback_index;
        const struct address_space_operations *a_ops;
        unsigned long           flags;
        errseq_t        wb_err;
        spinlock_t              private_lock;
        struct list_headprivate_list;
        void                    *private_data;
} __attribute__((aligned(sizeof(long)))) __randomize_layout;
        /*
         * On most architectures that alignment is already the case; but
         * must be enforced here for CRIS, to let the least significant bit
         * of struct page's "mapping" pointer be used for PAGE_MAPPING_ANON.
         */


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 */


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, bool spin);
        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;


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;


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

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

```
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 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;
};

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

struct delayed_work {
        struct work_struct work;
        struct timer_list timer;

        /* target workqueue and CPU ->timer uses to queue ->work */
        struct workqueue_struct *wq;
        int cpu;
};

enum {
        WQ_UNBOUND              = 1 << 1, /* not bound to any cpu */
        WQ_FREEZABLE            = 1 << 2, /* freeze during suspend */
        WQ_MEM_RECLAIM          = 1 << 3, /* may be used for memory reclaim */
        WQ_HIGHPRI              = 1 << 4, /* high priority */
        WQ_CPU_INTENSIVE= 1 << 5, /* cpu intensive workqueue */
        WQ_SYSFS        = 1 << 6, /* visible in sysfs, see
workqueue_sysfs_register() */

        /*
         * Per-cpu workqueues are generally preferred because they tend to
         * show better performance thanks to cache locality.  Per-cpu
         * workqueues exclude the scheduler from choosing the CPU to
         * execute the worker threads, which has an unfortunate side effect
         * of increasing power consumption.
         *
         * The scheduler considers a CPU idle if it doesn't have any task
         * to execute and tries to keep idle cores idle to conserve power;
         * however, for example, a per-cpu work item scheduled from an
         * interrupt handler on an idle CPU will force the scheduler to
         * execute the work item on that CPU breaking the idleness, which in
         * turn may lead to more scheduling choices which are sub-optimal
         * in terms of power consumption.
         *
         * Workqueues marked with WQ_POWER_EFFICIENT are per-cpu by default
         * but become unbound if workqueue.power_efficient kernel param is
         * specified.  Per-cpu workqueues which are identified to
         * contribute significantly to power-consumption are identified and
         * marked with this flag and enabling the power_efficient mode
         * leads to noticeable power saving at the cost of small
         * performance disadvantage.
         *
         * http://thread.gmane.org/gmane.linux.kernel/1480396
         */
        WQ_POWER_EFFICIENT      = 1 << 7,

        __WQ_DRAINING           = 1 << 16, /* internal: workqueue is draining */
        __WQ_ORDERED            = 1 << 17, /* internal: workqueue is ordered */
        __WQ_LEGACY             = 1 << 18, /* internal: create*_workqueue() */
        __WQ_ORDERED_EXPLICIT   = 1 << 19, /* internal: alloc_ordered_workqueue()
*/

        WQ_MAX_ACTIVE           = 512,    /* I like 512, better ideas? */
        WQ_MAX_UNBOUND_PER_CPU  = 4,      /* 4 * #cpus for unbound wq */
        WQ_DFL_ACTIVE           = WQ_MAX_ACTIVE / 2,
};

struct workqueue_struct {
        struct list_headpwqs;           /* WR: all pwqs of this wq */
        struct list_headlist;           /* 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_headflusher_queue;  /* WQ: flush waiters */
        struct list_headflusher_overflow; /* WQ: flush overflow list */

        struct list_headmaydays;/* 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 socket - general BSD socket
 *  @state: socket state (%SS_CONNECTED, etc)
 *  @type: socket type (%SOCK_STREAM, etc)
 *  @flags: socket flags (%SOCK_NOSPACE, etc)
 *  @ops: protocol specific socket operations
 *  @file: File back pointer for gc
 *  @sk: internal networking protocol agnostic socket representation
 *  @wq: wait queue for several uses
 */
struct socket {
        socket_state            state;
        short                   type;
        unsigned long           flags;
        struct file             *file;
        struct sock             *sk;
        const struct proto_ops  *ops;
        struct socket_wqwq;
};

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

/* Structure describing an Internet (IP) socket address. */
#if  __UAPI_DEF_SOCKADDR_IN
#define __SOCK_SIZE__   16              /* sizeof(struct sockaddr)      */
struct sockaddr_in {
  __kernel_sa_family_t  sin_family;     /* Address family               */
  __be16        sin_port;       /* Port number                  */
  struct in_addrsin_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)];
};
```