```
struct sock {
        struct sock_common      __sk_common; /* early demux fields */
        struct dst_entry __rcu  *sk_rx_dst;
        int                     sk_rx_dst_ifindex;
        u32                     sk_rx_dst_cookie;
        socket_lock_t           sk_lock;
        atomic_t                sk_drops;
        int                     sk_rcvlowat;
        struct sk_buff_head     sk_error_queue;
        struct sk_buff_head     sk_receive_queue;
        struct {
                atomic_t        rmem_alloc;
                int             len;
                struct sk_buff  *head;
                struct sk_buff  *tail;
        } sk_backlog;
        int                     sk_forward_alloc;
        u32                     sk_reserved_mem;
        #ifdef CONFIG_NET_RX_BUSY_POLL
        unsigned int            sk_ll_usec;
        /* ===== mostly read cache line ===== */
        unsigned int            sk_napi_id;
        #endif
        int                     sk_rcvbuf;
        int                     sk_disconnects;
        struct sk_filter __rcu  *sk_filter;
        union {
                struct socket_wq __rcu  *sk_wq;
                /* private: */
                struct socket_wq        *sk_wq_raw;
                /* public: */
        };
        #ifdef CONFIG_XFRM
        struct xfrm_policy __rcu *sk_policy[2];
        #endif
        struct dst_entry __rcu  *sk_dst_cache;
        atomic_t                sk_omem_alloc;
        int                     sk_sndbuf;
        /* ===== cache line for TX ===== */
        int                     sk_wmem_queued;
        refcount_t              sk_wmem_alloc;
        unsigned long           sk_tsq_flags;
        union {
                struct sk_buff  *sk_send_head;
                struct rb_root  tcp_rtx_queue;
        };
        struct sk_buff_head     sk_write_queue;
        __s32                   sk_peek_off;
        int                     sk_write_pending;
        __u32                   sk_dst_pending_confirm;
        u32                     sk_pacing_status; /* see enum sk_pacing */
        long                    sk_sndtimeo;
        struct timer_list       sk_timer;
        __u32                   sk_priority;
        __u32                   sk_mark;
        unsigned long           sk_pacing_rate; /* bytes per second */
        unsigned long           sk_max_pacing_rate;
        struct page_frag        sk_frag;
        netdev_features_t       sk_route_caps;
        int                     sk_gso_type;
        unsigned int            sk_gso_max_size;
        gfp_t                   sk_allocation;
        __u32                   sk_txhash;
        /*
        * Because of non atomicity rules, all
        * changes are protected by socket lock.
        */
        u8                      sk_gso_disabled : 1,
        sk_kern_sock : 1,
        sk_no_check_tx : 1,
```

```
        sk_no_check_rx : 1,
        sk_userlocks : 4;
        u8                      sk_pacing_shift;
        u16                     sk_type;
        u16                     sk_protocol;
        u16                     sk_gso_max_segs;
        unsigned long           sk_lingertime;
        struct proto            *sk_prot_creator;
        rwlock_t                sk_callback_lock;
        int                     sk_err,
        sk_err_soft;
        u32                     sk_ack_backlog;
        u32                     sk_max_ack_backlog;
        kuid_t                  sk_uid;
        u8                      sk_txrehash;
        #ifdef CONFIG_NET_RX_BUSY_POLL
        u8                      sk_prefer_busy_poll;
        u16                     sk_busy_poll_budget;
        #endif
        spinlock_t              sk_peer_lock;
        int                     sk_bind_phc;
        struct pid              *sk_peer_pid;
        const struct cred       *sk_peer_cred;
        long                    sk_rcvtimeo;
        ktime_t                 sk_stamp;
        #if BITS_PER_LONG==32
        seqlock_t               sk_stamp_seq;
        #endif
        atomic_t                sk_tskey;
        atomic_t                sk_zckey;
        u32                     sk_tsflags;
        u8                      sk_shutdown;
        u8                      sk_clockid;
        u8                      sk_txtime_deadline_mode : 1,
        sk_txtime_report_errors : 1,
        sk_txtime_unused : 6;
        bool                    sk_use_task_frag;
        struct socket           *sk_socket;
        void                    *sk_user_data;
        #ifdef CONFIG_SECURITY
        void                    *sk_security;
        #endif
        struct sock_cgroup_data sk_cgrp_data;
        struct mem_cgroup       *sk_memcg;
        void                    (*sk_state_change)(struct sock *sk);
        void                    (*sk_data_ready)(struct sock *sk);
        void                    (*sk_write_space)(struct sock *sk);
        void                    (*sk_error_report)(struct sock *sk);
        int                     (*sk_backlog_rcv)(struct sock *sk,
        struct sk_buff *skb);
        #ifdef CONFIG_SOCK_VALIDATE_XMIT
        struct sk_buff*         (*sk_validate_xmit_skb)(struct sock *sk,
        struct net_device *dev,
        struct sk_buff *skb);
        #endif
        void                    (*sk_destruct)(struct sock *sk);
        struct sock_reuseport __rcu     *sk_reuseport_cb;
        #ifdef CONFIG_BPF_SYSCALL
        struct bpf_local_storage __rcu  *sk_bpf_storage;
        #endif
        struct rcu_head         sk_rcu;
        netns_tracker           ns_tracker;
};
```

```
struct socket {
        socket_state            state;
        short                   type;
        unsigned long           flags;
        struct file             *file;
        struct sock             *sk;
        const struct proto_ops  *ops; /* Might change with IPV6_ADDRFORM or MPTCP. */
        struct socket_wq        wq;
};
```

```
struct sockaddr {
        sa_family_t     sa_family;      /* address family, AF_xxx       */
        char sa_data_min[14];
};

struct in_addr {
        u32 s_addr;
};

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)];
};
```

```
struct epoll_event {
        __poll_t events;
        __u64 data;
};
```

```
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 * const *s_xattr;
        #ifdef CONFIG_FS_ENCRYPTION
        const struct fscrypt_operations *s_cop;
        struct fscrypt_keyring  *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 bdev_handle      *s_bdev_handle;
        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;

        /*
        * 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;

        /* Read-only state of the superblock is being changed */
        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 *, enum freeze_holder who);
        int (*freeze_fs) (struct super_block *);
        int (*thaw_super) (struct super_block *, enum freeze_holder who);
        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 __rcu **(*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 *);
        void (*shutdown)(struct super_block *sb);
};
```

```
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 hlist_node d_sib;        /* child of parent list */
        struct hlist_head d_children;   /* 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;
};
```

```
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 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. */
        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; /* use inode_*_ctime accessors! */
        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_inode_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 mnt_idmap *, struct inode *, int);
        struct posix_acl * (*get_inode_acl)(struct inode *, int, bool);

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

        int (*create) (struct mnt_idmap *, struct inode *,struct dentry *,
        umode_t, bool);
        int (*link) (struct dentry *,struct inode *,struct dentry *);
        int (*unlink) (struct inode *,struct dentry *);
        int (*symlink) (struct mnt_idmap *, struct inode *,struct dentry *,
        const char *);
        int (*mkdir) (struct mnt_idmap *, struct inode *,struct dentry *,
        umode_t);
        int (*rmdir) (struct inode *,struct dentry *);
        int (*mknod) (struct mnt_idmap *, struct inode *,struct dentry *,
        umode_t,dev_t);
        int (*rename) (struct mnt_idmap *, struct inode *, struct dentry *,
        struct inode *, struct dentry *, unsigned int);
        int (*setattr) (struct mnt_idmap *, struct dentry *, struct iattr *);
        int (*getattr) (struct mnt_idmap *, 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 *, int);
        int (*atomic_open)(struct inode *, struct dentry *,
        struct file *, unsigned open_flag,
        umode_t create_mode);
        int (*tmpfile) (struct mnt_idmap *, struct inode *,
        struct file *, umode_t);
        struct posix_acl *(*get_acl)(struct mnt_idmap *, struct dentry *,
        int);
        int (*set_acl)(struct mnt_idmap *, struct dentry *,
        struct posix_acl *, int);
        int (*fileattr_set)(struct mnt_idmap *idmap,
        struct dentry *dentry, struct fileattr *fa);
        int (*fileattr_get)(struct dentry *dentry, struct fileattr *fa);
        struct offset_ctx *(*get_offset_ctx)(struct inode *inode);
} ____cacheline_aligned;
```

```
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;
};
```

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

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

```
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];
        unsigned long open_fds_init[1];
        unsigned long full_fds_bits_init[1];
        struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

```
struct file {
        union {
                /* fput() uses task work when closing and freeing file (default). */
                struct callback_head    f_task_work;
                /* fput() must use workqueue (most kernel threads). */
                struct llist_node       f_llist;
                unsigned int            f_iocb_flags;
        };
        /* Protects f_ep, f_flags. Must not be taken from IRQ context. */
        spinlock_t              f_lock;
        fmode_t                 f_mode;
        atomic_long_t           f_count;
        struct mutex            f_pos_lock;
        loff_t                  f_pos;
        unsigned int            f_flags;
        struct fown_struct      f_owner;
        const struct cred       *f_cred;
        struct file_ra_state    f_ra;
        struct path             f_path;
        struct inode            *f_inode;       /* cached value */
        const struct file_operations    *f_op;
        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)));
```

```
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 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_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 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_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 *);
        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);
        void (*splice_eof)(struct file *file);
        int (*setlease)(struct file *, int, 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);
        int (*uring_cmd)(struct io_uring_cmd *ioucmd, unsigned int issue_flags);
        int (*uring_cmd_iopoll)(struct io_uring_cmd *, struct io_comp_batch *,
        unsigned int poll_flags);
} __randomize_layout;
```

```
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;
};
```

```
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);
};
```

```
int single_open(struct file *file, int (*show)(struct seq_file *, void *),
void *data)
{
        struct seq_operations *op = kmalloc(sizeof(*op), GFP_KERNEL_ACCOUNT);
        int res = -ENOMEM;

        if (op) {
                op->start = single_start;
                op->next = single_next;
                op->stop = single_stop;
                op->show = show;
                res = seq_open(file, op);
                if (!res)
                ((struct seq_file *)file->private_data)->private = data;
                else
                kfree(op);
        }
        return res;
}
EXPORT_SYMBOL(single_open);
```

```
struct kmem_cache {
        #ifndef CONFIG_SLUB_TINY
        struct kmem_cache_cpu __percpu *cpu_slab;
        #endif
        /* Used for retrieving partial slabs, etc. */
        slab_flags_t flags;
        unsigned long min_partial;
        unsigned int size;              /* Object size including metadata */
        unsigned int object_size;       /* Object size without metadata */
        struct reciprocal_value reciprocal_size;
        unsigned int offset;            /* Free pointer offset */
        #ifdef CONFIG_SLUB_CPU_PARTIAL
        /* Number of per cpu partial objects to keep around */
        unsigned int cpu_partial;
        /* Number of per cpu partial slabs to keep around */
        unsigned int cpu_partial_slabs;
        #endif
        struct kmem_cache_order_objects oo;     /* Allocation and freeing of slabs */
        struct kmem_cache_order_objects min;
        gfp_t allocflags;               /* gfp flags to use on each alloc */
        int refcount;                   /* Refcount for slab cache destroy */
        void (*ctor)(void *object);     /* Object constructor */
        unsigned int inuse;             /* Offset to metadata */
        unsigned int align;             /* Alignment */
        unsigned int red_left_pad;      /* Left redzone padding size */
        const char *name;               /* Name (only for display!) */
        struct list_head list;          /* List of slab caches */
        #ifdef CONFIG_SYSFS
        struct kobject kobj;            /* For sysfs */
        #endif
        #ifdef CONFIG_SLAB_FREELIST_HARDENED
        unsigned long random;
        #endif
        #ifdef CONFIG_NUMA
        /** Defragmentation by allocating from a remote node. */
        unsigned int remote_node_defrag_ratio;
        #endif
        #ifdef CONFIG_SLAB_FREELIST_RANDOM
        unsigned int *random_seq;
        #endif
        #ifdef CONFIG_KASAN_GENERIC
        struct kasan_cache kasan_info;
        #endif
        #ifdef CONFIG_HARDENED_USERCOPY
        unsigned int useroffset;        /* Usercopy region offset */
        unsigned int usersize;          /* Usercopy region size */
        #endif
        struct kmem_cache_node *node[MAX_NUMNODES];
};
```

```
struct irqaction {
        irq_handler_t           handler;
        void                    *dev_id;
        void __percpu           *percpu_dev_id;
        struct irqaction        *next;
        irq_handler_t           thread_fn;
        struct task_struct      *thread;
        struct irqaction        *secondary;
        unsigned int            irq;
        unsigned int            flags;
        unsigned long           thread_flags;
        unsigned long           thread_mask;
        const char              *name;
        struct proc_dir_entry   *dir;
} ____cacheline_internodealigned_in_smp;
```

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

```
static struct softirq_action softirq_vec[NR_SOFTIRQS] __cacheline_aligned_in_smp;
const char * const softirq_to_name[NR_SOFTIRQS] = {
        "HI", "TIMER", "NET_TX", "NET_RX", "BLOCK", "IRQ_POLL",
        "TASKLET", "SCHED", "HRTIMER", "RCU"
};
```

```
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 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 __rcu **cpu_pwq; /* I: per-cpu pwqs */
};
```

```
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 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;
        const 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 */
        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
        #ifdef CONFIG_SWIOTLB_DYNAMIC
        struct list_head dma_io_tlb_pools;
        spinlock_t dma_io_tlb_lock;
        bool dma_uses_io_tlb;
        #endif
        struct dev_archdata     archdata; /* arch specific additions */
        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;
        const struct class      *class;
        const struct attribute_group **groups;  /* optional groups */
        void    (*release)(struct device *dev);
        struct iommu_group      *iommu_group;
        struct dev_iommu        *iommu;
        struct device_physical_location *physical_location;
        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;
        const 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;
};
```

```
struct usb_interface {
        /* array of alternate settings for this interface,
        * stored in no particular order */
        struct usb_host_interface *altsetting;

        struct usb_host_interface *cur_altsetting;      /* the currently
        * active alternate setting */
        unsigned num_altsetting;        /* number of alternate settings */

        /* If there is an interface association descriptor then it will list
        * the associated interfaces */
        struct usb_interface_assoc_descriptor *intf_assoc;

        int minor;                      /* minor number this interface is
        * bound to */
        enum usb_interface_condition condition;         /* state of binding */
        unsigned sysfs_files_created:1; /* the sysfs attributes exist */
        unsigned ep_devs_created:1;     /* endpoint "devices" exist */
        unsigned unregistering:1;       /* unregistration is in progress */
        unsigned needs_remote_wakeup:1; /* driver requires remote wakeup */
        unsigned needs_altsetting0:1;   /* switch to altsetting 0 is pending */
        unsigned needs_binding:1;       /* needs delayed unbind/rebind */
        unsigned resetting_device:1;    /* true: bandwidth alloc after reset */
        unsigned authorized:1;          /* used for interface authorization */
        enum usb_wireless_status wireless_status;
        struct work_struct wireless_status_work;

        struct device dev;              /* interface specific device info */
        struct device *usb_dev;
        struct work_struct reset_ws;    /* for resets in atomic context */
};
```

```
struct urb {
        /* private: usb core and host controller only fields in the urb */
        struct kref kref;               /* reference count of the URB */
        int unlinked;                   /* unlink error code */
        void *hcpriv;                   /* private data for host controller */
        atomic_t use_count;             /* concurrent submissions counter */
        atomic_t reject;                /* submissions will fail */
        /* public: documented fields in the urb that can be used by drivers */
        struct list_head urb_list;      /* list head for use by the urb's * current owner */
        struct list_head anchor_list;   /* the URB may be anchored */
        struct usb_anchor *anchor;
        struct usb_device *dev;         /* (in) pointer to associated device */
        struct usb_host_endpoint *ep;   /* (internal) pointer to endpoint */
        unsigned int pipe;              /* (in) pipe information */
        unsigned int stream_id;         /* (in) stream ID */
        int status;                     /* (return) non-ISO status */
        unsigned int transfer_flags;    /* (in) URB_SHORT_NOT_OK | ...*/
        void *transfer_buffer;          /* (in) associated data buffer */
        dma_addr_t transfer_dma;        /* (in) dma addr for transfer_buffer */
        struct scatterlist *sg;         /* (in) scatter gather buffer list */
        int num_mapped_sgs;             /* (internal) mapped sg entries */
        int num_sgs;                    /* (in) number of entries in the sg list */
        u32 transfer_buffer_length;     /* (in) data buffer length */
        u32 actual_length;              /* (return) actual transfer length */
        unsigned char *setup_packet;    /* (in) setup packet (control only) */
        dma_addr_t setup_dma;           /* (in) dma addr for setup_packet */
        int start_frame;                /* (modify) start frame (ISO) */
        int number_of_packets;          /* (in) number of ISO packets */
        int interval;                   /* (modify) transfer interval * (INT/ISO) */
        int error_count;                /* (return) number of ISO errors */
        void *context;                  /* (in) context for completion */
        usb_complete_t complete;        /* (in) completion routine */
        struct usb_iso_packet_descriptor iso_frame_desc[]; /* (in) ISO ONLY */
};
```

```
struct usb_driver {
        const char *name;

        int (*probe) (struct usb_interface *intf,
        const struct usb_device_id *id);

        void (*disconnect) (struct usb_interface *intf);

        int (*unlocked_ioctl) (struct usb_interface *intf, unsigned int code,
        void *buf);

        int (*suspend) (struct usb_interface *intf, pm_message_t message);
        int (*resume) (struct usb_interface *intf);
        int (*reset_resume)(struct usb_interface *intf);

        int (*pre_reset)(struct usb_interface *intf);
        int (*post_reset)(struct usb_interface *intf);

        const struct usb_device_id *id_table;
        const struct attribute_group **dev_groups;

        struct usb_dynids dynids;
        struct device_driver driver;
        unsigned int no_dynamic_id:1;
        unsigned int supports_autosuspend:1;
        unsigned int disable_hub_initiated_lpm:1;
        unsigned int soft_unbind:1;
};
```

```
struct usb_device_driver {
        const char *name;

        bool (*match) (struct usb_device *udev);
        int (*probe) (struct usb_device *udev);
        void (*disconnect) (struct usb_device *udev);

        int (*suspend) (struct usb_device *udev, pm_message_t message);
        int (*resume) (struct usb_device *udev, pm_message_t message);

        int (*choose_configuration) (struct usb_device *udev);

        const struct attribute_group **dev_groups;
        struct device_driver driver;
        const struct usb_device_id *id_table;
        unsigned int supports_autosuspend:1;
        unsigned int generic_subclass:1;
};
```

```
struct task_struct {
        #ifdef CONFIG_THREAD_INFO_IN_TASK
        struct thread_info thread_info;
        #endif
        unsigned int    __state;
        /* saved state for "spinlock sleepers" */
        unsigned int    saved_state;
        /*
        * This begins the randomizable portion of task_struct. Only
        * scheduling-critical items should be added above here.
        */
        randomized_struct_fields_start
        void*stack;
        refcount_t      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int    flags;
        unsigned int    ptrace;
        #ifdef CONFIG_MEM_ALLOC_PROFILING
        struct alloc_tag *alloc_tag;
        #endif
        #ifdef CONFIG_SMP
        int on_cpu;
        struct __call_single_node       wake_entry;
        unsigned int    wakee_flips;
        unsigned long   wakee_flip_decay_ts;
        struct task_struct *last_wakee;
        /*
        * recent_used_cpu is initially set as the last CPU used by a task
        * that wakes affine another task. Waker/wakee relationships can
        * push tasks around a CPU where each wakeup moves to the next one.
        * Tracking a recently used CPU allows a quick search for a recently
        * used CPU that may be idle.
        */
        int recent_used_cpu;
        int wake_cpu;
        #endif
        int on_rq;
        int prio;
        int static_prio;
        int normal_prio;
        unsigned int    rt_priority;
        struct sched_entityse;
        struct sched_rt_entity rt;
        struct sched_dl_entity dl;
        struct sched_dl_entity *dl_server;
        #ifdef CONFIG_SCHED_CLASS_EXT
        struct sched_ext_entity scx;
        #endif
        const struct sched_class        *sched_class;
        #ifdef CONFIG_SCHED_CORE
        struct rb_node  core_node;
        unsigned long   core_cookie;
        unsigned int    core_occupation;
        #endif
        #ifdef CONFIG_CGROUP_SCHED
        struct task_group *sched_task_group;
        #endif
        #ifdef CONFIG_UCLAMP_TASK
        /*
        * Clamp values requested for a scheduling entity.
        * Must be updated with task_rq_lock() held.
        */
        struct uclamp_se uclamp_req[UCLAMP_CNT];
        /*
        * Effective clamp values used for a scheduling entity.
        * Must be updated with task_rq_lock() held.
        */
        struct uclamp_se uclamp[UCLAMP_CNT];
        #endif
        struct sched_statistics         stats;
```

```
        #ifdef CONFIG_PREEMPT_NOTIFIERS
        /* List of struct preempt_notifier: */
        struct hlist_head preempt_notifiers;
        #endif
        #ifdef CONFIG_BLK_DEV_IO_TRACE
        unsigned int    btrace_seq;
        #endif
        unsigned int    policy;
        unsigned long   max_allowed_capacity;
        int nr_cpus_allowed;
        const cpumask_t *cpus_ptr;
        cpumask_t       *user_cpus_ptr;
        cpumask_t       cpus_mask;
        void *migration_pending;
        #ifdef CONFIG_SMP
        unsigned short  migration_disabled;
        #endif
        unsigned short  migration_flags;
        #ifdef CONFIG_PREEMPT_RCU
        int rcu_read_lock_nesting;
        union rcu_special rcu_read_unlock_special;
        struct list_head rcu_node_entry;
        struct rcu_node *rcu_blocked_node;
        #endif /* #ifdef CONFIG_PREEMPT_RCU */
        #ifdef CONFIG_TASKS_RCU
        unsigned long   rcu_tasks_nvcsw;
        u8rcu_tasks_holdout;
        u8rcu_tasks_idx;
        int rcu_tasks_idle_cpu;
        struct list_headrcu_tasks_holdout_list;
        int rcu_tasks_exit_cpu;
        struct list_head rcu_tasks_exit_list;
        #endif /* #ifdef CONFIG_TASKS_RCU */
        #ifdef CONFIG_TASKS_TRACE_RCU
        int trc_reader_nesting;
        int trc_ipi_to_cpu;
        union rcu_special trc_reader_special;
        struct list_head trc_holdout_list;
        struct list_head trc_blkd_node;
        int trc_blkd_cpu;
        #endif /* #ifdef CONFIG_TASKS_TRACE_RCU */
        struct sched_info sched_info;
        struct list_head tasks;
        #ifdef CONFIG_SMP
        struct plist_node pushable_tasks;
        struct rb_node  pushable_dl_tasks;
        #endif
        struct mm_struct *mm;
        struct mm_struct *active_mm;
        struct address_space *faults_disabled_mapping;
        int exit_state;
        int exit_code;
        int exit_signal;
        /* The signal sent when the parent dies: */
        int pdeath_signal;
        /* JOBCTL_*, siglock protected: */
        unsigned long   jobctl;
        /* Used for emulating ABI behavior of previous Linux versions: */
        unsigned int    personality;
        /* Scheduler bits, serialized by scheduler locks: */
        unsigned        sched_reset_on_fork:1;
        unsigned        sched_contributes_to_load:1;
        unsigned        sched_migrated:1;
        unsigned        sched_task_hot:1;
        /* Force alignment to the next boundary: */
        unsigned        :0;
        /* Unserialized, strictly 'current' */
        /*
        * This field must not be in the scheduler word above due to wakelist
        * queueing no longer being serialized by p->on_cpu. However:
        * p->XXX = X;   ttwu()
```

```
142         * schedule()      if (p->on_rq && ..) // false
143         *   smp_mb__after_spinlock();     if (smp_load_acquire(&p->on_cpu) && //true
144         *   deactivate_task()      ttwu_queue_wakelist())
145         *     p->on_rq = 0;     p->sched_remote_wakeup = Y;
146         * guarantees all stores of 'current' are visible before
147         * ->sched_remote_wakeup gets used, so it can be in this word.
148         */
149        unsigned        sched_remote_wakeup:1;
150        #ifdef CONFIG_RT_MUTEXES
151        unsigned        sched_rt_mutex:1;
152        #endif
153        /* Bit to tell TOMOYO we're in execve(): */
154        unsigned        in_execve:1;
155        unsigned        in_iowait:1;
156        #ifndef TIF_RESTORE_SIGMASK
157        unsigned        restore_sigmask:1;
158        #endif
159        #ifdef CONFIG_MEMCG_V1
160        unsigned        in_user_fault:1;
161        #endif
162        #ifdef CONFIG_LRU_GEN
163        /* whether the LRU algorithm may apply to this access */
164        unsigned        in_lru_fault:1;
165        #endif
166        #ifdef CONFIG_COMPAT_BRK
167        unsigned        brk_randomized:1;
168        #endif
169        #ifdef CONFIG_CGROUPS
170        /* disallow userland-initiated cgroup migration */
171        unsigned        no_cgroup_migration:1;
172        /* task is frozen/stopped (used by the cgroup freezer) */
173        unsigned        frozen:1;
174        #endif
175        #ifdef CONFIG_BLK_CGROUP
176        unsigned        use_memdelay:1;
177        #endif
178        #ifdef CONFIG_PSI
179        /* Stalled due to lack of memory */
180        unsigned        in_memstall:1;
181        #endif
182        #ifdef CONFIG_PAGE_OWNER
183        /* Used by page_owner=on to detect recursion in page tracking. */
184        unsigned        in_page_owner:1;
185        #endif
186        #ifdef CONFIG_EVENTFD
187        /* Recursion prevention for eventfd_signal() */
188        unsigned        in_eventfd:1;
189        #endif
190        #ifdef CONFIG_ARCH_HAS_CPU_PASID
191        unsigned        pasid_activated:1;
192        #endif
193        #ifdef CONFIG_X86_BUS_LOCK_DETECT
194        unsigned        reported_split_lock:1;
195        #endif
196        #ifdef CONFIG_TASK_DELAY_ACCT
197        /* delay due to memory thrashing */
198        unsigned                        in_thrashing:1;
199        #endif
200        #ifdef CONFIG_PREEMPT_RT
201        struct netdev_xmitnet_xmit;
202        #endif
203        unsigned long   atomic_flags; /* Flags requiring atomic access. */
204        struct restart_blockrestart_block;
205        pid_t pid;
206        pid_t tgid;
207        #ifdef CONFIG_STACKPROTECTOR
208        /* Canary value for the -fstack-protector GCC feature: */
209        unsigned long   stack_canary;
210        #endif
211        /*
212        * Point ers to the (original) parent process, youngest child, younger sibling,
```

```
213         * older sibling, respectively.  (p->father can be replaced with
214         * p->real_parent->pid)
215         */
216         /* Real parent process: */
217         struct task_struct __rcu        *real_parent;
218         /* Recipient of SIGCHLD, wait4() reports: */
219         struct task_struct __rcu        *parent;
220         /*
221         * Children/sibling form the list of natural children:
222         */
223         struct list_head children;
224         struct list_head sibling;
225         struct task_struct *group_leader;
226         /*
227         * 'ptraced' is the list of tasks this task is using ptrace() on.
228         * This includes both natural children and PTRACE_ATTACH targets.
229         * 'ptrace_entry' is this task's link on the p->parent->ptraced list.
230         */
231         struct list_head ptraced;
232         struct list_head ptrace_entry;
233         /* PID/PID hash table linkage. */
234         struct pid      *thread_pid;
235         struct hlist_nodepid_links[PIDTYPE_MAX];
236         struct list_head thread_node;
237         struct completion *vfork_done;
238         /* CLONE_CHILD_SETTID: */
239         int  __user     *set_child_tid;
240         /* CLONE_CHILD_CLEARTID: */
241         int  __user     *clear_child_tid;
242         /* PF_KTHREAD | PF_IO_WORKER */
243         void * worker_private;
244         u64 utime;
245         u64 stime;
246         #ifdef CONFIG_ARCH_HAS_SCALED_CPUTIME
247         u64 utimescaled;
248         u64  stimescaled;
249         #endif
250         u64  gtime;
251         struct prev_cpu timeprev_cputime;
252         #ifdef CONFIG_VIRT_CPU_ACCOUNTING_GEN
253         struct vtime    vtime;
254         #endif
255         #ifdef CONFIG_NO_HZ_FULL
256         atomic_t        tick_dep_mask;
257         #endif
258         /* Context switch counts: */
259         unsigned long   nvcsw;
260         unsigned long   nivcsw;
261         /* Monotonic time in nsecs: */
262         u64 start_time;
263         /* Boot based time in nsecs: */
264         u64 start_boottime;
265         /* MM fault and swap info: this can arguably be seen as either mm-specific or
    thread-specific: */
266         unsigned long   min_flt;
267         unsigned long   maj_flt;
268         /* Empty if CONFIG_POSIX_CPUTIMERS=n */
269         struct posix_cputimersposix_cputimers;
270         #ifdef CONFIG_POSIX_CPU_TIMERS_TASK_WORK
271         struct posix_cputimers_work     posix_cputimers_work;
272         #endif
273         /* Process credentials: */
274         /* Tracer's credentials at attach: */
275         const struct cred __rcu*ptracer_cred;
276         /* Objective and real subjective task credentials (COW): */
277         const struct cred __rcu*real_cred;
278         /* Effective (overridable) subjective task credentials (COW): */
279         const struct cred __rcu*cred;
280         #ifdef CONFIG_KEYS
281         /* Cached requested key. */
282         struct key      *cached_requested_key;
```

```
283         #endif
284         /*
285         * executable name, excluding path.
286         * - normally initialized begin_new_exec()
287         * - set it with set_task_comm()
288         *   - strscpy_pad() to ensure it is always NUL-terminated and zero-padded
289         *   - task_lock() to ensure the operation is atomic and the name is fully updated.
290         */
291         char comm[TASK_COMM_LEN];
292         struct nameidata*nameidata;
293         #ifdef CONFIG_SYSVIPC
294         struct sysv_sem sysvsem;
295         struct sysv_shm sysvshm;
296         #endif
297         #ifdef CONFIG_DETECT_HUNG_TASK
298         unsigned long   last_switch_count;
299         unsigned long   last_switch_time;
300         #endif
301         /* Filesystem information: */
302         struct fs_struct *fs;
303         /* Open file information: */
304         struct files_struct *files;
305         #ifdef CONFIG_IO_URING
306         struct io_uring_task*io_uring;
307         #endif
308         /* Namespaces: */
309         struct nsproxy  *nsproxy;
310         /* Signal handlers: */
311         struct signal_struct *signal;
312         struct sighand_struct __rcu*sighand;
313         sigset_t        blocked;
314         sigset_t        real_blocked;
315         /* Restored if set_restore_sigmask() was used: */
316         sigset_t        saved_sigmask;
317         struct sigpendingpending;
318         unsigned long   sas_ss_sp;
319         size_tsas_ss_size;
320         unsigned int    sas_ss_flags;
321         struct callback_head *task_works;
322         #ifdef CONFIG_AUDIT
323         #ifdef CONFIG_AUDITSYSCALL
324         struct audit_context *audit_context;
325         #endif
326         kuid_t loginuid;
327         unsigned int    sessionid;
328         #endif
329         struct seccomp  seccomp;
330         struct syscall_user_dispatch    syscall_dispatch;
331         /* Thread group tracking: */
332         u64  parent_exec_id;
333         u64  self_exec_id;
334         /* Protection against (de-)allocation: mm, files, fs, tty, keyrings, mems_allowed,
    mempolicy: */
335         spinlock_t      alloc_lock;
336         /* Protection of the PI data structures: */
337         raw_spinlock_t  pi_lock;
338         struct wake_q_nodewake_q;
339         #ifdef CONFIG_RT_MUTEXES
340         /* PI waiters blocked on a rt_mutex held by this task: */
341         struct rb_root_cachedpi_waiters;
342         /* Updated under owner's pi_lock and rq lock */
343         struct task_struct *pi_top_task;
344         /* Deadlock detection and priority inheritance handling: */
345         struct rt_mutex_waiter *pi_blocked_on;
346         #endif
347         #ifdef CONFIG_DEBUG_MUTEXES
348         /* Mutex deadlock detection: */
349         struct mutex_waiter *blocked_on;
350         #endif
351         #ifdef CONFIG_DETECT_HUNG_TASK_BLOCKER
352         struct mutex    *blocker_mutex;
```

```
353         #endif
354         #ifdef CONFIG_DEBUG_ATOMIC_SLEEP
355         int non_block_count;
356         #endif
357         #ifdef CONFIG_TRACE_IRQFLAGS
358         struct irqtrace_eventsirqtrace;
359         unsigned int    hardirq_threaded;
360         u64 hardirq_chain_key;
361         int softirqs_enabled;
362         int softirq_context;
363         int irq_config;
364         #endif
365         #ifdef CONFIG_PREEMPT_RT
366         int softirq_disable_cnt;
367         #endif
368         #ifdef CONFIG_LOCKDEP
369         # define MAX_LOCK_DEPTH 48UL
370         u64 curr_chain_key;
371         int lockdep_depth;
372         unsigned int    lockdep_recursion;
373         struct held_lockheld_locks[MAX_LOCK_DEPTH];
374         #endif
375         #if defined(CONFIG_UBSAN) && !defined(CONFIG_UBSAN_TRAP)
376         unsigned int    in_ubsan;
377         #endif
378         /* Journalling filesystem info: */
379         void * journal_info;
380         /* Stacked block device info: */
381         struct bio_list *bio_list;
382         /* Stack plugging: */
383         struct blk_plug *plug;
384         /* VM state: */
385         struct reclaim_state*reclaim_state;
386         struct io_context*io_context;
387         #ifdef CONFIG_COMPACTION
388         struct capture_control*capture_control;
389         #endif
390         /* Ptrace state: */
391         unsigned long   ptrace_message;
392         kernel_siginfo_t*last_siginfo;
393         struct task_io_accounting       ioac;
394         #ifdef CONFIG_PSI
395         /* Pressure stall state */
396         unsigned int    psi_flags;
397         #endif
398         #ifdef CONFIG_TASK_XACCT
399         /* Accumulated RSS usage: */
400         u64 acct_rss_mem1;
401         /* Accumulated virtual memory usage: */
402         u64 acct_vm_mem1;
403         /* stime + utime since last update: */
404         u64 acct_timexpd;
405         #endif
406         #ifdef CONFIG_CPUSETS
407         /* Protected by ->alloc_lock: */
408         nodemask_t      mems_allowed;
409         /* Sequence number to catch updates: */
410         seqcount_spinlock_tmems_allowed_seq;
411         int cpuset_mem_spread_rotor;
412         #endif
413         #ifdef CONFIG_CGROUPS
414         /* Control Group info protected by css_set_lock: */
415         struct css_set __rcu *cgroups;
416         /* cg_list protected by css_set_lock and tsk->alloc_lock: */
417         struct list_head cg_list;
418         #endif
419         #ifdef CONFIG_X86_CPU_RESCTRL
420         u32 closid;
421         u32 rmid;
422         #endif
423         #ifdef CONFIG_FUTEX
```

```
424         struct robust_list_head __user  *robust_list;
425         #ifdef CONFIG_COMPAT
426         struct compat_robust_list_head __user *compat_robust_list;
427         #endif
428         struct list_head pi_state_list;
429         struct futex_pi_state *pi_state_cache;
430         struct mutex    futex_exit_mutex;
431         unsigned int    futex_state;
432         #endif
433         #ifdef CONFIG_PERF_EVENTS
434         u8perf_recursion[PERF_NR_CONTEXTS];
435         struct perf_event_context       *perf_event_ctxp;
436         struct mutex    perf_event_mutex;
437         struct list_headperf_event_list;
438         struct perf_ctx_data __rcu      *perf_ctx_data;
439         #endif
440         #ifdef CONFIG_DEBUG_PREEMPT
441         unsigned long   preempt_disable_ip;
442         #endif
443         #ifdef CONFIG_NUMA
444         /* Protected by alloc_lock: */
445         struct mempolicy*mempolicy;
446         short il_prev;
447         u8 il_weight;
448         short pref_node_fork;
449         #endif
450         #ifdef CONFIG_NUMA_BALANCING
451         int numa_scan_seq;
452         unsigned int    numa_scan_period;
453         unsigned int    numa_scan_period_max;
454         int numa_preferred_nid;
455         unsigned long   numa_migrate_retry;
456         /* Migration stamp: */
457         u64 node_stamp;
458         u64 last_task_numa_placement;
459         u64 last_sum_exec_runtime;
460         struct callback_headnuma_work;
461         /*
462         * This point er is only modified for current in syscall and
463         * pagefault context (and for tasks being destroyed), so it can be read
464         * from any of the following contexts:
465         *  - RCU read-side critical section
466         *  - current->numa_group from everywhere
467         *  - task's runqueue locked, task not running
468         */
469         struct numa_group __rcu* numa_group;
470         /*
471         * numa_faults is an array split int o four regions:
472         * faults_memory, faults_cpu, faults_memory_buffer, faults_cpu_buffer
473         * in this precise order.
474         * faults_memory: Exponential decaying average of faults on a per-node
475         * basis. Scheduling placement decisions are made based on these
476         * counts. The values remain static for the duration of a PTE scan.
477         * faults_cpu: Track the nodes the process was running on when a NUMA
478         * hint ing fault was incurred.
479         * faults_memory_buffer and faults_cpu_buffer: Record faults per node
480         * during the current scan window. When the scan completes, the counts
481         * in faults_memory and faults_cpu decay and these values are copied.
482         */
483         unsigned long   *numa_faults;
484         unsigned long   total_numa_faults;
485         /*
486         * numa_faults_locality tracks if faults recorded during the last
487         * scan window were remote/local or failed to migrate. The task scan
488         * period is adapted based on the locality of the faults with different
489         * weights depending on whether they were shared or private faults
490         */
491         unsigned long   numa_faults_locality[3];
492         unsigned long   numa_pages_migrated;
493         #endif /* CONFIG_NUMA_BALANCING */
494         #ifdef CONFIG_RSEQ
```

```
        struct rseq __user *rseq;
        u32 rseq_len;
        u32 rseq_sig;
        /*
        * RmW on rseq_event_mask must be performed atomically
        * with respect to preemption.
        */
        unsigned long rseq_event_mask;
        # ifdef CONFIG_DEBUG_RSEQ
        /*
        * This is a place holder to save a copy of the rseq fields for
        * validation of read-only fields. The struct rseq has a
        * variable-length array at the end, so it cannot be used
        * directly. Reserve a size large enough for the known fields.
        */
        char rseq_fields[sizeof(struct rseq)];
        # endif
        #endif
        #ifdef CONFIG_SCHED_MM_CID
        int mm_cid;/* Current cid in mm */
        int last_mm_cid;        /* Most recent cid in mm */
        int migrate_from_cpu;
        int mm_cid_active;      /* Whether cid bitmap is active */
        struct callback_headcid_work;
        #endif
        struct tlbflush_unmap_batch     tlb_ubc;
        /* Cache last used pipe for splice(): */
        struct pipe_inode_info*splice_pipe;
        struct page_fragtask_frag;
        #ifdef CONFIG_TASK_DELAY_ACCT
        struct task_delay_info*delays;
        #endif
        #ifdef CONFIG_FAULT_INJECTION
        int make_it_fail;
        unsigned int    fail_nth;
        #endif
        /*
        * When (nr_dirtied >= nr_dirtied_pause), it's time to call
        * balance_dirty_pages() for a dirty throttling pause:
        */
        int nr_dirtied;
        int nr_dirtied_pause;
        /* Start of a write-and-pause period: */
        unsigned long   dirty_paused_when;
        #ifdef CONFIG_LATENCYTOP
        int latency_record_count;
        struct latency_recordlatency_record[LT_SAVECOUNT];
        #endif
        /*
        * Time slack values; these are used to round up poll() and
        * select() etc timeout values. These are in nanoseconds.
        */
        u64 timer_slack_ns;
        u64 default_timer_slack_ns;
        #if defined(CONFIG_KASAN_GENERIC) || defined(CONFIG_KASAN_SW_TAGS)
        unsigned int    kasan_depth;
        #endif
        #ifdef CONFIG_KCSAN
        struct kcsan_ctxkcsan_ctx;
        #ifdef CONFIG_TRACE_IRQFLAGS
        struct irqtrace_eventskcsan_save_irqtrace;
        #endif
        #ifdef CONFIG_KCSAN_WEAK_MEMORY
        int  kcsan_stack_depth;
        #endif
        #endif
        #ifdef CONFIG_KMSAN
        struct kmsan_ctxkmsan_ctx;
        #endif
        #if IS_ENABLED(CONFIG_KUNIT)
        struct kunit    *kunit_test;
```

```
566         #endif
567         #ifdef CONFIG_FUNCTION_GRAPH_TRACER
568         /* Index of current stored address in ret_stack: */
569         int  curr_ret_stack;
570         int  curr_ret_depth;
571         /* Stack of return addresses for return function tracing: */
572         unsigned long   *ret_stack;
573         /* Timestamp for last schedule: */
574         unsigned long longftrace_timestamp;
575         unsigned long longftrace_sleeptime;
576         /*
577         * Number of functions that haven't been traced
578         * because of depth overrun:
579         */
580         atomic_t        trace_overrun;
581         /* Pause tracing: */
582         atomic_t        tracing_graph_pause;
583         #endif
584         #ifdef CONFIG_TRACING
585         /* Bitmask and counter of trace recursion: */
586         unsigned long   trace_recursion;
587         #endif /* CONFIG_TRACING */
588         #ifdef CONFIG_KCOV
589         /* See kernel/kcov.c for more details. */
590         /* Coverage collection mode enabled for this task (0 if disabled): */
591         unsigned int    kcov_mode;
592         /* Size of the kcov_area: */
593         unsigned int    kcov_size;
594         /* Buffer for coverage collection: */
595         void * kcov_area;
596         /* KCOV descriptor wired with this task or NULL: */
597         struct kcov     *kcov;
598         /* KCOV common handle for remote coverage collection: */
599         u64 kcov_handle;
600         /* KCOV sequence number: */
601         int  kcov_sequence;
602         /* Collect coverage from softirq context: */
603         unsigned int    kcov_softirq;
604         #endif
605         #ifdef CONFIG_MEMCG_V1
606         struct mem_cgroup *memcg_in_oom;
607         #endif
608         #ifdef CONFIG_MEMCG
609         /* Number of pages to reclaim on returning to userland: */
610         unsigned int    memcg_nr_pages_over_high;
611         /* Used by memcontrol for targeted memcg charge: */
612         struct mem_cgroup*active_memcg;
613         /* Cache for current->cgroups->memcg->objcg lookups: */
614         struct obj_cgroup *objcg;
615         #endif
616         #ifdef CONFIG_BLK_CGROUP
617         struct gendisk  *throttle_disk;
618         #endif
619         #ifdef CONFIG_UPROBES
620         struct uprobe_task *utask;
621         #endif
622         #if defined(CONFIG_BCACHE) || defined(CONFIG_BCACHE_MODULE)
623         unsigned int    sequential_io;
624         unsigned int    sequential_io_avg;
625         #endif
626         struct kmap_ctrlkmap_ctrl;
627         #ifdef CONFIG_DEBUG_ATOMIC_SLEEP
628         unsigned long   task_state_change;
629         # ifdef CONFIG_PREEMPT_RT
630         unsigned long   saved_state_change;
631         # endif
632         #endif
633         struct rcu_head rcu;
634         refcount_t      rcu_users;
635         int  pagefault_disabled;
636         #ifdef CONFIG_MMU
```

```
        struct task_struct *oom_reaper_list;
        struct timer_list oom_reaper_timer;
        #endif
        #ifdef CONFIG_VMAP_STACK
        struct vm_struct *stack_vm_area;
        #endif
        #ifdef CONFIG_THREAD_INFO_IN_TASK
        /* A live task holds one reference: */
        refcount_t      stack_refcount;
        #endif
        #ifdef CONFIG_LIVEPATCH
        int   patch_state;
        #endif
        #ifdef CONFIG_SECURITY
        /* Used by LSM modules for access restriction: */
        void * security;
        #endif
        #ifdef CONFIG_BPF_SYSCALL
        /* Used by BPF task local storage */
        struct bpf_local_storage __rcu  *bpf_storage;
        /* Used for BPF run context */
        struct bpf_run_ctx*bpf_ctx;
        #endif
        /* Used by BPF for per-TASK xdp storage */
        struct bpf_net_context*bpf_net_context;
        #ifdef CONFIG_GCC_PLUGIN_STACKLEAK
        unsigned long   lowest_stack;
        unsigned long   prev_lowest_stack;
        #endif
        #ifdef CONFIG_X86_MCE
        void __user     *mce_vaddr;
        __u64 mce_kflags;
        u64 mce_addr;
        __u64 mce_ripv : 1,
        mce_whole_page : 1,
        __mce_reserved : 62;
        struct callback_headmce_kill_me;
        int  mce_count;
        #endif
        #ifdef CONFIG_KRETPROBES
        struct llist_head               kretprobe_instances;
        #endif
        #ifdef CONFIG_RETHOOK
        struct llist_head               rethooks;
        #endif
        #ifdef CONFIG_ARCH_HAS_PARANOID_L1D_FLUSH
        /*
        * If L1D flush is supported on mm context switch
        * then we use this callback head to queue kill work
        * to kill tasks that are not running on SMT disabled
        * cores
        */
        struct callback_headl1d_flush_kill;
        #endif
        #ifdef CONFIG_RV
        /*
        * Per-task RV monitor. Nowadays fixed in RV_PER_TASK_MONITORS.
        * If we find justification for more monitors, we can think
        * about adding more or developing a dynamic method. So far,
        * none of these are justified.
        */
        union rv_task_monitorrv[RV_PER_TASK_MONITORS];
        #endif
        #ifdef CONFIG_USER_EVENTS
                struct user_event_mm *user_event_mm;
        #endif
        randomized_struct_fields_end
        /* CPU-specific state of this task: */
        struct thread_struct thread;
};
```