

СОДЕРЖАНИЕ

1	Структуры VFS и доп.	2
2	Тасклеты	11
3	Очереди работ	12
4	USB	16
5	proc	19
6	seq	21
7	Буф. ввод-вывод	22

1 Структуры VFS и доп.

Листинг 1.1 – Структура struct super_block

```
1 struct super_block { // v 4.10
2     struct list_head    s_list;        /* Keep this first */
3
4     unsigned long       s_blocksize;
5     loff_t               s_maxbytes; /* Max file size */
6     struct file_system_type *s_type;
7     const struct super_operations *s_op;
8
9     unsigned long       s_flags;
10
11    unsigned long        s_magic;
12    struct dentry         *s_root;
13    struct rw_semaphore s_umount;
14    int                   s_count;
15
16    char s_id[32];        /* Informational name */
17
18    const struct dentry_operations *s_d_op; /* default d_op for
19        dentries */
20
21    /*
22     * Keep the lru lists last in the structure so they always
23     * sit on their
24     * own individual cachelines.
25     */
26    struct list_lru       s_dentry_lru
27        ____cacheline_aligned_in_smp;
28    struct list_lru       s_inode_lru ____cacheline_aligned_in_smp;
29
30    struct list_head      s_inodes;     /* all inodes */
31 };
```

Листинг 1.2 – Структура struct super_operations

```
1 struct super_operations { // v 4.10
2     struct inode *(*alloc_inode)(struct super_block *sb);
3     void (*destroy_inode)(struct inode *);
4 }
```

```

5 void (*dirty_inode) (struct inode *, int flags);
6 int (*write_inode) (struct inode *, struct writeback_control
    *wbc);
7 int (*drop_inode) (struct inode *);
8 void (*evict_inode) (struct inode *);
9 void (*put_super) (struct super_block *);
10 int (*sync_fs)(struct super_block *sb, int wait);
11 int (*freeze_super) (struct super_block *);
12 int (*freeze_fs) (struct super_block *);
13 int (*thaw_super) (struct super_block *);
14 int (*unfreeze_fs) (struct super_block *);
15 int (*statfs) (struct dentry *, struct kstatfs *);
16 int (*remount_fs) (struct super_block *, int *, char *);
17 void (*umount_begin) (struct super_block *);
18
19 int (*show_options)(struct seq_file *, struct dentry *);
20 int (*show_devname)(struct seq_file *, struct dentry *);
21 int (*show_path)(struct seq_file *, struct dentry *);
22 int (*show_stats)(struct seq_file *, struct dentry *);
23 #ifdef CONFIG_QUOTA
24     ssize_t (*quota_read)(struct super_block *, int, char *,
        size_t, loff_t);
25     ssize_t (*quota_write)(struct super_block *, int, const char
        *, size_t, loff_t);
26     struct dquot **(*get_dquots)(struct inode *);
27 #endif
28 int (*bdev_try_to_free_page)(struct super_block *, struct
    page *, gfp_t);
29 long (*nr_cached_objects)(struct super_block *,
30     struct shrink_control *);
31 long (*free_cached_objects)(struct super_block *,
32     struct shrink_control *);
33 };

```

Листинг 1.3 – Структура struct inode

```

1 struct inode {
2     umode_t          i_mode;
3     kuid_t           i_uid;
4     kgid_t           i_gid;
5     unsigned int      i_flags;
6
7     const struct inode_operations *i_op;

```

```

8      struct super_block  *i_sb;
9
10     /* Stat data, not accessed from path walking */
11     unsigned long        i_ino;
12     /*
13      * Filesystems may only read i_nlink directly.  They shall
14      * use the
15      * following functions for modification:
16      *
17      * (set|clear|inc|drop)_nlink
18      * inode_(inc|dec)_link_count
19      */
20     union {
21         const unsigned int i_nlink;
22         unsigned int __i_nlink;
23     };
24     dev_t                i_rdev;
25     loff_t               i_size;
26     struct timespec      i_atime;
27     struct timespec      i_mtime;
28     struct timespec      i_ctime;
29     spinlock_t           i_lock; /* i_blocks, i_bytes, maybe i_size */
30     unsigned short       i_bytes;
31     unsigned int          i_blkbits;
32     blkcnt_t             i_blocks;
33     u64                  i_version;
34     atomic_t             i_count;
35 };

```

Листинг 1.4 – Структура struct inode_operations

```

1 struct inode_operations {
2     struct dentry * (*lookup) (struct inode *, struct dentry *,
3     unsigned int);
4     const char * (*get_link) (struct dentry *, struct inode *,
5     struct delayed_call *);
6     int (*permission) (struct inode *, int);
7     struct posix_acl * (*get_acl)(struct inode *, int);
8
9     int (*readlink) (struct dentry *, char __user *, int);
10
11     int (*create) (struct inode *, struct dentry *, umode_t,
12     bool);

```

```

10  int (*link) (struct dentry *, struct inode *, struct dentry *);
11  int (*unlink) (struct inode *, struct dentry *);
12  int (*symlink) (struct inode *, struct dentry *, const char *);
13  int (*mkdir) (struct inode *, struct dentry *, umode_t);
14  int (*rmdir) (struct inode *, struct dentry *);
15  int (*mknod) (struct inode *, struct dentry *, umode_t, dev_t);
16  int (*rename) (struct inode *, struct dentry *,
17                struct inode *, struct dentry *, unsigned int);
18  int (*setattr) (struct dentry *, struct iattr *);
19  int (*getattr) (struct vfsmount *mnt, struct dentry *,
20                struct kstat *);
21  ssize_t (*listxattr) (struct dentry *, char *, size_t);
22  int (*fiemap) (struct inode *, struct fiemap_extents_info *,
23                u64 start,
24                u64 len);
25  int (*update_time) (struct inode *, struct timespec *, int);
26  int (*atomic_open) (struct inode *, struct dentry *,
27                    struct file *, unsigned open_flag,
28                    umode_t create_mode, int *opened);
29  int (*tmpfile) (struct inode *, struct dentry *, umode_t);
30  int (*set_acl) (struct inode *, struct posix_acl *, int);
31 } ____cacheline_aligned;

```

Листинг 1.5 – Структура struct dentry

```

1  struct dentry {
2      /* RCU lookup touched fields */
3      unsigned int d_flags;          /* protected by d_lock */
4      seqcount_t d_seq;              /* per dentry seqlock */
5      struct hlist_bl_node d_hash;    /* lookup hash list */
6      struct dentry *d_parent;        /* parent directory */
7      struct qstr d_name;
8      struct inode *d_inode;          /* Where the name belongs to -
9      NULL is
10      * negative */
11      unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
12      /* Ref lookup also touches following */
13      struct lockref d_lockref; /* per-dentry lock and refcount
14      */
15      const struct dentry_operations *d_op;
16      struct super_block *d_sb; /* The root of the dentry tree */
17      unsigned long d_time; /* used by d_revalidate */

```

```

17     void *d_fsdata;           /* fs-specific data */
18
19     union {
20         struct list_head d_lru;      /* LRU list */
21         wait_queue_head_t *d_wait;  /* in-lookup ones only */
22     };
23     struct list_head d_child; /* child of parent list */
24     struct list_head d_subdirs; /* our children */
25     /*
26      * d_alias and d_rcu can share memory
27      */
28     union {
29         struct hlist_node d_alias; /* inode alias list */
30         struct hlist_bl_node d_in_lookup_hash; /* only for
31             in-lookup ones */
32         struct rcu_head d_rcu;
33     } d_u;
34 };

```

Листинг 1.6 – Структура struct dentry_operations

```

1 struct dentry_operations {
2     int (*d_revalidate)(struct dentry *, unsigned int);
3     int (*d_weak_revalidate)(struct dentry *, unsigned int);
4     int (*d_hash)(const struct dentry *, struct qstr *);
5     int (*d_compare)(const struct dentry *,
6         unsigned int, const char *, const struct qstr *);
7     int (*d_delete)(const struct dentry *);
8     int (*d_init)(struct dentry *);
9     void (*d_release)(struct dentry *);
10    void (*d_prune)(struct dentry *);
11    void (*d_iput)(struct dentry *, struct inode *);
12    char *(*d_dname)(struct dentry *, char *, int);
13    struct vfsmount *(*d_automount)(struct path *);
14    int (*d_manage)(const struct path *, bool);
15    struct dentry *(*d_real)(struct dentry *, const struct inode
16        *,
17        unsigned int);
18 } ____cacheline_aligned;

```

Листинг 1.7 – Структура struct file

```

1 struct file {
2     union {

```

```

3         struct llist_node    fu_llist;
4         struct rcu_head      fu_rcuhead;
5     } f_u;
6     struct path              f_path;
7     struct inode             *f_inode;    /* cached value */
8     const struct file_operations *f_op;
9
10    /*
11     * Protects f_ep_links, f_flags.
12     * Must not be taken from IRQ context.
13     */
14    spinlock_t                f_lock;
15    atomic_long_t              f_count;
16    unsigned int               f_flags;
17    fmode_t                    f_mode;
18    struct mutex                f_pos_lock;
19    loff_t                     f_pos;
20    struct fown_struct          f_owner;
21    const struct cred           *f_cred;
22    struct file_ra_state        f_ra;
23
24    u64                        f_version;
25 #ifdef CONFIG_SECURITY
26     void                      *f_security;
27 #endif
28     /* needed for tty driver, and maybe others */
29     void                      *private_data;
30
31 #ifdef CONFIG_EPOLL
32     /* Used by fs/eventpoll.c to link all the hooks to this file
33      */
34     struct list_head          f_ep_links;
35     struct list_head          f_tfile_llink;
36 #endif /* #ifdef CONFIG_EPOLL */
37     struct address_space       *f_mapping;
38 } __attribute__((aligned(4))); /* lest something weird decides
    that 2 is OK */

```

Листинг 1.8 – Структура struct file_operations

```

1 struct file_operations {
2     struct module *owner;
3     loff_t (*llseek) (struct file *, loff_t, int);

```

```

4      ssize_t (*read) (struct file *, char __user *, size_t,
      loff_t *);
5      ssize_t (*write) (struct file *, const char __user *,
      size_t, loff_t *);
6      ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
7      ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
8      int (*iterate) (struct file *, struct dir_context *);
9      int (*iterate_shared) (struct file *, struct dir_context *);
10     unsigned int (*poll) (struct file *, struct
      poll_table_struct *);
11     long (*unlocked_ioctl) (struct file *, unsigned int,
      unsigned long);
12     long (*compat_ioctl) (struct file *, unsigned int, unsigned
      long);
13     int (*mmap) (struct file *, struct vm_area_struct *);
14     int (*open) (struct inode *, struct file *);
15     int (*flush) (struct file *, fl_owner_t id);
16     int (*release) (struct inode *, struct file *);
17     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
18     int (*fasync) (int, struct file *, int);
19     int (*lock) (struct file *, int, struct file_lock *);
20     ssize_t (*sendpage) (struct file *, struct page *, int,
      size_t, loff_t *, int);
21     unsigned long (*get_unmapped_area)(struct file *, unsigned
      long, unsigned long, unsigned long, unsigned long);
22     int (*check_flags)(int);
23     int (*flock) (struct file *, int, struct file_lock *);
24     ssize_t (*splice_write)(struct pipe_inode_info *, struct
      file *, loff_t *, size_t, unsigned int);
25     ssize_t (*splice_read)(struct file *, loff_t *, struct
      pipe_inode_info *, size_t, unsigned int);
26     int (*setlease)(struct file *, long, struct file_lock **,
      void **);
27     long (*fallocate)(struct file *file, int mode, loff_t offset,
      loff_t len);
28     void (*show_fdinfo)(struct seq_file *m, struct file *f);
29 #ifndef CONFIG_MMU
30     unsigned (*mmap_capabilities)(struct file *);
31 #endif
32     ssize_t (*copy_file_range)(struct file *, loff_t, struct
      file *,

```



```

34         loff_t, size_t, unsigned int);
35     int (*clone_file_range)(struct file *, loff_t, struct file
        *, loff_t,
36         u64);
37     ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct
        file *,
38         u64);
39 };

```

Листинг 1.9 – Структура struct file_system_type

```

1 struct file_system_type {
2     const char *name;
3     int fs_flags;
4 #define FS_REQUIRES_DEV      1
5 #define FS_BINARY_MOUNTDATA  2
6 #define FS_HAS_SUBTYPE       4
7 #define FS_USERNS_MOUNT      8    /* Can be mounted by userns root
    */
8 #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle
    d_move() during rename() internally. */
9     struct dentry *(*mount) (struct file_system_type *, int,
        const char *, void *);
10    void (*kill_sb) (struct super_block *);
11    struct module *owner;
12    struct file_system_type * next;
13    struct hlist_head fs_supers;
14
15
16    struct lock_class_key s_lock_key;
17    struct lock_class_key s_umount_key;
18    struct lock_class_key s_vfs_rename_key;
19    struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];
20
21    struct lock_class_key i_lock_key;
22    struct lock_class_key i_mutex_key;
23    struct lock_class_key i_mutex_dir_key;
24 };

```

Листинг 1.10 – Структура struct fs_struct

```

1 struct fs_struct {
2     int users;
3     spinlock_t lock;
4     seqcount_t seq;

```

```

5     int umask;
6     int in_exec;
7     struct path root, pwd;
8 };

```

Листинг 1.11 – Структура struct files_struct

```

1  /*
2   * Open file table structure
3   */
4  struct files_struct {
5      /*
6       * read mostly part
7       */
8      atomic_t count;
9      bool resize_in_progress;
10     wait_queue_head_t resize_wait;
11
12     struct fdtable __rcu *fdt;
13     struct fdtable fdtab;
14     /*
15      * written part on a separate cache line in SMP
16      */
17     spinlock_t file_lock ____cacheline_aligned_in_smp;
18     unsigned int next_fd;
19     unsigned long close_on_exec_init[1];
20     unsigned long open_fds_init[1];
21     unsigned long full_fds_bits_init[1];
22     struct file __rcu * fd_array[NR_OPEN_DEFAULT];
23 };

```

Листинг 1.12 – Структура struct path

```

1  struct path {
2     struct vfsmount *mnt;
3     struct dentry *dentry;
4 };

```

2 Тасклеты

Листинг 2.1 – Структура struct tasklet_struct

```
1 struct tasklet_struct
2 {
3     struct tasklet_struct *next;
4     unsigned long state;
5     atomic_t count;
6     void (*func)(unsigned long);
7     unsigned long data;
8 };
```

3 Очереди работ

Листинг 3.1 – Структура struct workqueue_struct

```
1  /*
2   * The externally visible workqueue. It relays the issued work
3   * items to
4   * the appropriate worker_pool through its pool_workqueues.
5   */
6  struct workqueue_struct {
7      struct list_head    pwqs;          /* WR: all pwqs of this wq */
8      struct list_head    list;          /* PR: list of all
9                                          workqueues */
10
11      struct mutex        mutex;          /* protects this wq */
12      int                 work_color;     /* WQ: current work color */
13      int                 flush_color;    /* WQ: current flush color */
14      atomic_t            nr_pwqs_to_flush; /* flush in progress */
15      struct wq_flusher   *first_flusher; /* WQ: first flusher */
16      struct list_head    flusher_queue;  /* WQ: flush waiters */
17      struct list_head    flusher_overflow; /* WQ: flush overflow
18                                          list */
19
20      struct list_head    maydays;       /* MD: pwqs requesting
21                                          rescue */
22      struct worker        *rescuer;      /* I: rescue worker */
23
24      int                 nr_drainers;     /* WQ: drain in progress */
25      int                 saved_max_active; /* WQ: saved pwq max_active */
26
27      struct workqueue_attrs *unbound_attrs; /* PW: only for
28                                          unbound wqs */
29      struct pool_workqueue *dfl_pwq;     /* PW: only for unbound
30                                          wqs */
31
32      #ifdef CONFIG_SYSFS
33          struct wq_device *wq_dev;       /* I: for sysfs interface */
34      #endif
35      #ifdef CONFIG_LOCKDEP
36          struct lockdep_map lockdep_map;
37      #endif
38      char                name[WQ_NAME_LEN]; /* I: workqueue name */
39  }
```

```

33
34     /*
35      * Destruction of workqueue_struct is sched-RCU protected to
36      * allow
37      * walking the workqueues list without grabbing
38      * wq_pool_mutex.
39      * This is used to dump all workqueues from sysrq.
40      */
41 struct rcu_head      rcu;
42
43 /* hot fields used during command issue, aligned to
44    cacheline */
45 unsigned int        flags ____cacheline_aligned; /* WQ: WQ_*
46    flags */
47 struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs
48    */
49 struct pool_workqueue __rcu *numa_pwq_tbl[]; /* PWR: unbound
50    pwqs indexed by node */
51 };

```

Листинг 3.2 – Структура struct work_struct

```

1 struct work_struct {
2     atomic_long_t data;
3     struct list_head entry;
4     work_func_t func;
5 #ifdef CONFIG_LOCKDEP
6     struct lockdep_map lockdep_map;
7 #endif
8 };

```

Листинг 3.3 – Структура struct pool_workqueue

```

1 struct pool_workqueue {
2     struct worker_pool *pool;          /* I: the associated pool */
3     struct workqueue_struct *wq;       /* I: the owning
4     workqueue */
5     int work_color; /* L: current color */
6     int flush_color; /* L: flushing color */
7     int refcnt; /* L: reference count */
8     int nr_in_flight[WORK_NR_COLORS];
9     /* L: nr of in_flight works */
10    int nr_active; /* L: nr of active works */
11    int max_active; /* L: max active works */

```

```

11     struct list_head    delayed_works;    /* L: delayed works */
12     struct list_head    pwqs_node;    /* WR: node on wq->pwqs */
13     struct list_head    mayday_node;    /* MD: node on
        wq->maydays */
14
15     /*
16      * Release of unbound pwq is punted to system_wq. See
        put_pwq()
17      * and pwq_unbound_release_workfn() for details.
        pool_workqueue
18      * itself is also sched-RCU protected so that the first pwq
        can be
19      * determined without grabbing wq->mutex.
20      */
21     struct work_struct    unbound_release_work;
22     struct rcu_head        rcu;
23 } __aligned(1 << WORK_STRUCT_FLAG_BITS);

```

Листинг 3.4 – Структура struct worker_pool

```

1 struct worker_pool {
2     spinlock_t        lock;    /* the pool lock */
3     int                cpu;    /* I: the associated cpu */
4     int                node;    /* I: the associated node ID */
5     int                id;    /* I: pool ID */
6     unsigned int        flags;    /* X: flags */
7
8     unsigned long        watchdog_ts;    /* L: watchdog timestamp
        */
9
10    struct list_head    worklist;    /* L: list of pending works
        */
11    int                nr_workers;    /* L: total number of workers */
12
13    /* nr_idle includes the ones off idle_list for rebinding */
14    int                nr_idle;    /* L: currently idle ones */
15
16    struct list_head    idle_list;    /* X: list of idle workers */
17    struct timer_list    idle_timer;    /* L: worker idle timeout */
18    struct timer_list    mayday_timer;    /* L: SOS timer for
        workers */
19
20    /* a workers is either on busy_hash or idle_list, or the

```

```

    manager */
21 DECLARE_HASHTABLE(busy_hash, BUSY_WORKER_HASH_ORDER);
22     /* L: hash of busy workers */
23
24 /* see manage_workers() for details on the two manager
    mutexes */
25 struct mutex      manager_arb;    /* manager arbitration */
26 struct worker     *manager;      /* L: purely informational */
27 struct mutex      attach_mutex;   /* attach/detach
    exclusion */
28 struct list_head   workers;       /* A: attached workers */
29 struct completion  *detach_completion; /* all workers
    detached */
30
31 struct ida         worker_ida; /* worker IDs for task name */
32
33 struct workqueue_attrs *attrs;     /* I: worker attributes
    */
34 struct hlist_node   hash_node; /* PL: unbound_pool_hash
    node */
35 int                refcnt;        /* PL: refcnt for unbound pools */
36
37 /*
38  * The current concurrency level. As it's likely to be
    accessed
39  * from other CPUs during try_to_wake_up(), put it in a
    separate
40  * cacheline.
41  */
42 atomic_t           nr_running ____cacheline_aligned_in_smp;
43
44 /*
45  * Destruction of pool is sched-RCU protected to allow
    dereferences
46  * from get_work_pool().
47  */
48 struct rcu_head     rcu;
49 } ____cacheline_aligned_in_smp;

```

4 USB

Листинг 4.1 – Структура struct device_driver

```
1 struct device_driver {
2     const char      *name;
3     struct bus_type  *bus;
4
5     struct module    *owner;
6     const char      *mod_name; /* used for built-in modules */
7
8     bool suppress_bind_attrs; /* disables bind/unbind via
9                               sysfs */
10
11     enum probe_type probe_type;
12
13
14     const struct of_device_id *of_match_table;
15     const struct acpi_device_id *acpi_match_table;
16
17     int (*probe) (struct device *dev);
18     int (*remove) (struct device *dev);
19     void (*shutdown) (struct device *dev);
20     int (*suspend) (struct device *dev, pm_message_t state);
21     int (*resume) (struct device *dev);
22     const struct attribute_group **groups;
23
24     const struct dev_pm_ops *pm;
25
26     struct driver_private *p;
27 };
```

Листинг 4.2 – Структура struct device

```
1 struct device {
2     struct device      *parent;
3     struct kobject kobj;
4     const char      *init_name; /* initial name of the device */
5     const struct device_type *type;
6
7     struct bus_type *bus; /* type of bus device is on */
8     struct device_driver *driver; /* which driver has
9                                   allocated this
10                                   device */
11 };
```



```

11 #ifdef CONFIG_NUMA
12     int        numa_node;  /* NUMA node this device is close to */
13 #endif
14     u64        *dma_mask;  /* dma mask (if dma'able device) */
15
16     dev_t      devt;       /* dev_t, creates the sysfs "dev" */
17
18 };

```

Листинг 4.3 – Структура struct usb_driver

```

1 struct usb_driver {
2     const char *name;
3
4     int (*probe) (struct usb_interface *intf,
5                  const struct usb_device_id *id);
6
7     void (*disconnect) (struct usb_interface *intf);
8
9     int (*unlocked_ioctl) (struct usb_interface *intf, unsigned
10                          int code,
11                          void *buf);
12
13     int (*suspend) (struct usb_interface *intf, pm_message_t
14                   message);
15
16     int (*resume) (struct usb_interface *intf);
17     int (*reset_resume)(struct usb_interface *intf);
18
19     int (*pre_reset)(struct usb_interface *intf);
20     int (*post_reset)(struct usb_interface *intf);
21
22     const struct usb_device_id *id_table;
23
24     struct usb_dynids dynids;
25     struct usbdrv_wrap drvwrap;
26     unsigned int no_dynamic_id:1;
27     unsigned int supports_autosuspend:1;
28     unsigned int disable_hub_initiated_lpm:1;
29     unsigned int soft_unbind:1;
30 };

```

Листинг 4.4 – Структура struct usb_device_driver

```

1 struct usb_device_driver {

```

```
2     const char *name;
3
4     int (*probe) (struct usb_device *udev);
5     void (*disconnect) (struct usb_device *udev);
6
7     int (*suspend) (struct usb_device *udev, pm_message_t
        message);
8     int (*resume) (struct usb_device *udev, pm_message_t
        message);
9     struct usbdrv_wrap drvwrap;
10    unsigned int supports_autosuspend:1;
11 };
```

5 proc

Листинг 5.1 – Структура struct proc_dir_entry

```
1 struct proc_dir_entry {
2     /*
3      * number of callers into module in progress;
4      * negative -> it's going away RSN
5      */
6     atomic_t in_use;
7     refcount_t refcnt;
8     struct list_head pde_openers;    /* who did ->open, but not
9      * ->release */
10    /* protects ->pde_openers and all struct pde_opener
11     * instances */
12    spinlock_t pde_unload_lock;
13    struct completion *pde_unload_completion;
14    const struct inode_operations *proc_iops;
15    union {
16        const struct proc_ops *proc_ops;
17        const struct file_operations *proc_dir_ops;
18    };
19    const struct dentry_operations *proc_dops;
20    union {
21        const struct seq_operations *seq_ops;
22        int (*single_show)(struct seq_file *, void *);
23    };
24    proc_write_t write;
25    void *data;
26    unsigned int state_size;
27    unsigned int low_ino;
28    nlink_t nlink;
29    kuid_t uid;
30    kgid_t gid;
31    loff_t size;
32    struct proc_dir_entry *parent;
33    struct rb_root subdir;
34    struct rb_node subdir_node;
35    char *name;
36    umode_t mode;
37    u8 flags;
38    u8 namelen;
```

```
37     char inline_name[];
38 } __randomize_layout;
```

Листинг 5.2 – Структура struct proc_ops

```
1 struct proc_ops {
2     unsigned int proc_flags;
3     int (*proc_open)(struct inode *, struct file *);
4     ssize_t (*proc_read)(struct file *, char __user *, size_t,
5         loff_t *);
6     ssize_t (*proc_read_iter)(struct kiocb *, struct iov_iter *);
7     ssize_t (*proc_write)(struct file *, const char __user *,
8         size_t, loff_t *);
9     /* mandatory unless nonseekable_open() or equivalent is used
10        */
11     loff_t (*proc_lseek)(struct file *, loff_t, int);
12     int (*proc_release)(struct inode *, struct file *);
13     __poll_t (*proc_poll)(struct file *, struct
14         poll_table_struct *);
15     long (*proc_ioctl)(struct file *, unsigned int, unsigned
16         long);
17 #ifdef CONFIG_COMPAT
18     long (*proc_compat_ioctl)(struct file *, unsigned int,
19         unsigned long);
20 #endif
21     int (*proc_mmap)(struct file *, struct vm_area_struct *);
22     unsigned long (*proc_get_unmapped_area)(struct file *,
23         unsigned long, unsigned long, unsigned long, unsigned
24         long);
25 } __randomize_layout;
```

6 seq

Листинг 6.1 – Структура structs seq_file

```
1 struct seq_file {
2     char *buf;
3     size_t size;
4     size_t from;
5     size_t count;
6     size_t pad_until;
7     loff_t index;
8     loff_t read_pos;
9     struct mutex lock;
10    const struct seq_operations *op;
11    int poll_event;
12    const struct file *file;
13    void *private;
14 };
15
16 struct seq_operations {
17     void * (*start) (struct seq_file *m, loff_t *pos);
18     void (*stop) (struct seq_file *m, void *v);
19     void * (*next) (struct seq_file *m, void *v, loff_t *pos);
20     int (*show) (struct seq_file *m, void *v);
21 };
```

7 Буф. ввод-вывод

Листинг 7.1 – Структура struct _IO_FILE

```
1 typedef struct _IO_FILE FILE;
2
3 struct _IO_FILE
4 {
5     int _flags;          /* High-order word is _IO_MAGIC; rest is
6                          flags. */
7
8     /* The following pointers correspond to the C++ streambuf
9      protocol. */
10    char *_IO_read_ptr;   /* Current read pointer */
11    char *_IO_read_end;   /* End of get area. */
12    char *_IO_read_base;  /* Start of putback+get area. */
13    char *_IO_write_base; /* Start of put area. */
14    char *_IO_write_ptr;  /* Current put pointer. */
15    char *_IO_write_end;  /* End of put area. */
16    char *_IO_buf_base;   /* Start of reserve area. */
17    char *_IO_buf_end;    /* End of reserve area. */
18
19    /* The following fields are used to support backing up and
20     undo. */
21    char *_IO_save_base; /* Pointer to start of non-current get
22                          area. */
23    char *_IO_backup_base; /* Pointer to first valid character of
24                          backup area */
25    char *_IO_save_end; /* Pointer to end of non-current get area.
26                          */
27
28    struct _IO_marker *_markers;
29
30    struct _IO_FILE *_chain;
31
32    int _fileno;
33    int _flags2;
34    __off_t _old_offset; /* This used to be _offset but it's too
35                          small. */
36    /* 1+column number of pbase(); 0 is unknown. */
37    unsigned short _cur_column;
38    signed char _vtable_offset;
```

```

32     char _shortbuf[1];
33
34     _IO_lock_t *_lock;
35 #ifdef _IO_USE_OLD_IO_FILE
36 };

```

Листинг 7.2 – Структура struct stat

```

1 struct stat {
2     unsigned long st_dev;      /* Device. */
3     unsigned long st_ino;      /* File serial number. */
4     unsigned int  st_mode;     /* File mode. */
5     unsigned int  st_nlink;    /* Link count. */
6     unsigned int  st_uid;      /* User ID of the file's owner. */
7     unsigned int  st_gid;      /* Group ID of the file's group. */
8     unsigned long st_rdev;     /* Device number, if device. */
9     unsigned long __pad1;
10    long          st_size;      /* Size of file, in bytes. */
11    int           st_blksize;   /* Optimal block size for I/O. */
12    int           __pad2;
13    long          st_blocks;    /* Number 512-byte blocks allocated. */
14    long          st_atime;     /* Time of last access. */
15    unsigned long st_atime_nsec;
16    long          st_mtime;     /* Time of last modification. */
17    unsigned long st_mtime_nsec;
18    long          st_ctime;     /* Time of last status change. */
19    unsigned long st_ctime_nsec;
20    unsigned int  __unused4;
21    unsigned int  __unused5;
22 };

```