СОДЕРЖАНИЕ

1 Структуры VFS и доп.

Листинг 1.1 – Структура struct super_block

```
struct super_block { // v 4.10
    struct list_head    s_list;     /* Keep this first */

    unsigned long       s_blocksize;
    loff_t          s_maxbytes; /* Max file size */
    struct file_system_type *s_type;
    const struct super_operations   *s_op;

    unsigned long       s_flags;

    unsigned long       s_magic;
    struct dentry       *s_root;
    struct rw_semaphore s_umount;
    int         s_count;

    char s_id[32];              /* Informational name */

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

    /*
     * Keep the lru lists last in the structure so they always
        sit on their
     * own individual cachelines.
     */
    struct list_lru     s_dentry_lru
       ____cacheline_aligned_in_smp;
    struct list_lru     s_inode_lru ____cacheline_aligned_in_smp;

    struct list_head    s_inodes;   /* all inodes */

};
```

Листинг 1.2 – Структура struct super_operations

```
struct super_operations { // v 4.10
    struct inode *(*alloc_inode)(struct super_block *sb);
    void (*destroy_inode)(struct inode *);

```

```
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
            use the
14       * following functions for modification:
15       *
16       *    (set|clear|inc|drop)_nlink
17       *    inode_(inc|dec)_link_count
18       */
19      union {
20          const unsigned int i_nlink;
21          unsigned int __i_nlink;
22      };
23      dev_t           i_rdev;
24      loff_t          i_size;
25      struct timespec     i_atime;
26      struct timespec     i_mtime;
27      struct timespec     i_ctime;
28      spinlock_t      i_lock; /* i_blocks, i_bytes, maybe i_size */
29      unsigned short          i_bytes;
30      unsigned int        i_blkbits;
31      blkcnt_t        i_blocks;
32      u64         i_version;
33      atomic_t        i_count;
34  };
```

Листинг 1.4 – Структура struct inode_operations

```
1   struct inode_operations {
2       struct dentry * (*lookup) (struct inode *,struct dentry *,
          unsigned int);
3       const char * (*get_link) (struct dentry *, struct inode *,
          struct delayed_call *);
4       int (*permission) (struct inode *, int);
5       struct posix_acl * (*get_acl)(struct inode *, int);
6
7       int (*readlink) (struct dentry *, char __user *,int);
8
9       int (*create) (struct inode *,struct dentry *, umode_t,
          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 *,
          struct kstat *);
20      ssize_t (*listxattr) (struct dentry *, char *, size_t);
21      int (*fiemap)(struct inode *, struct fiemap_extent_info *,
          u64 start,
22              u64 len);
23      int (*update_time)(struct inode *, struct timespec *, int);
24      int (*atomic_open)(struct inode *, struct dentry *,
25                  struct file *, unsigned open_flag,
26                  umode_t create_mode, int *opened);
27      int (*tmpfile) (struct inode *, struct dentry *, umode_t);
28      int (*set_acl)(struct inode *, struct posix_acl *, int);
29  } ____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 -
          NULL is
9                           * negative */
10      unsigned char d_iname[DNAME_INLINE_LEN];    /* small names */
11
12      /* Ref lookup also touches following */
13      struct lockref d_lockref;   /* per-dentry lock and refcount
          */
14      const struct dentry_operations *d_op;
15      struct super_block *d_sb;   /* The root of the dentry tree */
16      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
              in-lookup ones */
31          struct rcu_head d_rcu;
32      } d_u;
33  };
```

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                       unsigned int);
17  } ____cacheline_aligned;
```

Листинг 1.7 – Структура struct file

```
1   struct file {
2       union {
```

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

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

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

#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the hooks to this file
       */
    struct list_head    f_ep_links;
    struct list_head    f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space    *f_mapping;
} __attribute__((aligned(4)));  /* lest something weird decides
   that 2 is OK */
```

Листинг 1.8 – Структура struct file_operations

```
struct file_operations {
    struct module *owner;
    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,
28                loff_t len);
29      void (*show_fdinfo)(struct seq_file *m, struct file *f);
30  #ifndef CONFIG_MMU
31      unsigned (*mmap_capabilities)(struct file *);
32  #endif
33      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,
10                  const char *, void *);
11      void (*kill_sb) (struct super_block *);
12      struct module *owner;
13      struct file_system_type * next;
14      struct hlist_head fs_supers;
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;
```

```
    int umask;
    int in_exec;
    struct path root, pwd;
};
```

Листинг 1.11 – Структура struct files_struct

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

    struct fdtable __rcu *fdt;
    struct fdtable fdtab;
  /*
   * written part on a separate cache line in SMP
   */
    spinlock_t file_lock ____cacheline_aligned_in_smp;
    unsigned int next_fd;
    unsigned long close_on_exec_init[1];
    unsigned long open_fds_init[1];
    unsigned long full_fds_bits_init[1];
    struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

Листинг 1.12 – Структура struct path

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

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

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

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

    struct list_head    maydays;    /* MD: pwqs requesting
       rescue */
    struct worker       *rescuer;   /* I: 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
    struct lockdep_map  lockdep_map;
#endif
    char            name[WQ_NAME_LEN]; /* I: workqueue name */
```

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

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
           workqueue */
4      int         work_color; /* L: current color */
5      int         flush_color;    /* L: flushing color */
6      int         refcnt;     /* L: reference count */
7      int         nr_in_flight[WORK_NR_COLORS];
8                          /* L: nr of in_flight works */
9      int         nr_active;  /* L: nr of active works */
10     int         max_active; /* L: max active works */
```

```
    struct list_head    delayed_works;  /* L: delayed works */
    struct list_head    pwqs_node;  /* WR: node on wq->pwqs */
    struct list_head    mayday_node;    /* MD: node on
       wq->maydays */

    /*
     * Release of unbound pwq is punted to system_wq.  See
        put_pwq()
     * and pwq_unbound_release_workfn() for details.
        pool_workqueue
     * itself is also sched-RCU protected so that the first pwq
        can be
     * determined without grabbing wq->mutex.
     */
    struct work_struct  unbound_release_work;
    struct rcu_head     rcu;
} __aligned(1 << WORK_STRUCT_FLAG_BITS);
```

Листинг 3.4 – Структура struct worker_pool

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

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

    struct list_head    worklist;   /* L: list of pending works
       */
    int         nr_workers; /* L: total number of workers */

    /* nr_idle includes the ones off idle_list for rebinding */
    int         nr_idle;    /* L: currently idle ones */

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

    /* a workers is either on busy_hash or idle_list, or the
```

```
        manager */
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

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

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

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

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

    int (*probe) (struct device *dev);
    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 dev_pm_ops *pm;

    struct driver_private *p;
};
```

Листинг 4.2 – Структура struct device

```
struct device {
    struct device       *parent;
    struct kobject kobj;
    const char      *init_name; /* initial name of the device */
    const struct device_type *type;

    struct bus_type *bus;       /* type of bus device is on */
    struct device_driver *driver;   /* which driver has
       allocated this
                       device */

```

```
#ifdef CONFIG_NUMA
    int     numa_node;  /* NUMA node this device is close to */
#endif
    u64     *dma_mask;  /* dma mask (if dma'able device) */

    dev_t           devt;   /* dev_t, creates the sysfs "dev" */

};
```

Листинг 4.3 – Структура struct usb_driver

```
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;

    struct usb_dynids dynids;
    struct usbdrv_wrap drvwrap;
    unsigned int no_dynamic_id:1;
    unsigned int supports_autosuspend:1;
    unsigned int disable_hub_initiated_lpm:1;
    unsigned int soft_unbind:1;
};
```

Листинг 4.4 – Структура struct usb_device_driver

```
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
           ->release */
9      /* protects ->pde_openers and all struct pde_opener
           instances */
10     spinlock_t pde_unload_lock;
11     struct completion *pde_unload_completion;
12     const struct inode_operations *proc_iops;
13     union {
14         const struct proc_ops *proc_ops;
15         const struct file_operations *proc_dir_ops;
16     };
17     const struct dentry_operations *proc_dops;
18     union {
19         const struct seq_operations *seq_ops;
20         int (*single_show)(struct seq_file *, void *);
21     };
22     proc_write_t write;
23     void *data;
24     unsigned int state_size;
25     unsigned int low_ino;
26     nlink_t nlink;
27     kuid_t uid;
28     kgid_t gid;
29     loff_t size;
30     struct proc_dir_entry *parent;
31     struct rb_root subdir;
32     struct rb_node subdir_node;
33     char *name;
34     umode_t mode;
35     u8 flags;
36     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,
            loff_t *);
5       ssize_t (*proc_read_iter)(struct kiocb *, struct iov_iter *);
6       ssize_t (*proc_write)(struct file *, const char __user *,
            size_t, loff_t *);
7       /* mandatory unless nonseekable_open() or equivalent is used
            */
8       loff_t  (*proc_lseek)(struct file *, loff_t, int);
9       int (*proc_release)(struct inode *, struct file *);
10      __poll_t (*proc_poll)(struct file *, struct
            poll_table_struct *);
11      long    (*proc_ioctl)(struct file *, unsigned int, unsigned
            long);
12  #ifdef CONFIG_COMPAT
13      long    (*proc_compat_ioctl)(struct file *, unsigned int,
            unsigned long);
14  #endif
15      int (*proc_mmap)(struct file *, struct vm_area_struct *);
16      unsigned long (*proc_get_unmapped_area)(struct file *,
            unsigned long, unsigned long, unsigned long, unsigned
            long);
17  } __randomize_layout;
```

6 seq

Листинг 6.1 – Структура structs seq_file

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

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);
};
```

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

```
  char _shortbuf[1];

  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

Листинг 7.2 – Структура struct stat

```
struct stat {
  unsigned long st_dev;     /* Device.  */
  unsigned long st_ino;     /* File serial number.  */
  unsigned int  st_mode;    /* File mode.  */
  unsigned int  st_nlink;   /* Link count.  */
  unsigned int  st_uid;     /* User ID of the file's owner.  */
  unsigned int  st_gid;     /* Group ID of the file's group. */
  unsigned long st_rdev;    /* Device number, if device.  */
  unsigned long __pad1;
  long       st_size;    /* Size of file, in bytes.  */
  int        st_blksize; /* Optimal block size for I/O.  */
  int        __pad2;
  long       st_blocks;  /* Number 512-byte blocks allocated. */
  long       st_atime;   /* Time of last access.  */
  unsigned long st_atime_nsec;
  long       st_mtime;   /* Time of last modification.  */
  unsigned long st_mtime_nsec;
  long       st_ctime;   /* Time of last status change.  */
  unsigned long st_ctime_nsec;
  unsigned int  __unused4;
  unsigned int  __unused5;
};
```