1. Коды ядра

Прерывания

```
struct cdev {
  ...
  struct module *owner;
  const struct file_operations *ops;
  struct list_head list;
  dev_t dev;
  ...
} __randomize_layout;
```

```
struct block_device {
  dev_t               bd_dev;
  ...
  struct inode *          bd_inode;
  struct super_block *    bd_super;
  ...
  struct gendisk *    bd_disk; // (определяет устройство (специализирова
      нный интерфейс), ссылается на struct block_device_operations)
  ...
} __randomize_layout;
```

```
struct tasklet_struct
{
  struct tasklet_struct *next; // указатель на следующий тасклет в однос
      вязном списке
  unsigned long state; // состояние тасклета
  atomic_t count; // счетчик ссылок
  bool use_callback; // флаг
  union {
    void (*func)(unsigned long data); // функция-обработчик тасклета
    void (*callback)(struct tasklet_struct *t);
  };
  unsigned long data; // аргумент функции-обработчика тасклет
};
```

```
/*
* Очередь отложенных действий, связанная с процессором:
*/
struct cpu_workqueue_struct
{
    spinlock_t lock; /* Очередь для защиты данной структуры */
    long remove_sequence; /* последний добавленный элемент
   (следующий для запуска ) */
    long insert_sequence; /* следующий элемент для добавления */
    struct list_head worklist; /* список действий на каждое cpu */
    wait_queue_head_t more_work;
    wait_queue_head_t work_done;
   struct workqueue_struct *wq; /* соответствующая структура
                  workqueue_struct */
   task_t *thread; /* соответствующий поток (функция) */
   int run_depth; /* глубина рекурсии функции run_workqueue() */
};
```

```
struct workqueue_struct {
struct list_head   pwqs;    /* WR: all pwqs of this wq */
struct list_head   list;    /* PR: list of all workqueues */

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

struct list_head   maydays;  /* MD: pwqs requesting rescue */
struct worker    *rescuer; /* MD: rescue worker */

int       nr_drainers;  /* WQ: drain in progress */
int       saved_max_active; /* WQ: saved pwq max_active */

struct workqueue_attrs  *unbound_attrs; /* PW: only for unbound wqs */
```

```
  struct pool_workqueue *dfl_pwq; /* PW: only for unbound wqs */

#ifdef CONFIG_SYSFS
  struct wq_device  *wq_dev;  /* I: for sysfs interface */
#endif
#ifdef CONFIG_LOCKDEP
  char        *lock_name;
  struct lock_class_key key;
  struct lockdep_map  lockdep_map;
#endif
  char         name[WQ_NAME_LEN]; /* I: workqueue name */

  /*
   * Destruction of workqueue_struct is RCU protected to allow walking
   * the workqueues list without grabbing wq_pool_mutex.
   * This is used to dump all workqueues from sysrq.
   */
  struct rcu_head    rcu;

  /* hot fields used during command issue, aligned to cacheline */
  unsigned int     flags ____cacheline_aligned; /* WQ: WQ_* flags */
  struct pool_workqueue __percpu *cpu_pwqs; /* I: per-cpu pwqs */
  struct pool_workqueue __rcu *numa_pwq_tbl[]; /* PWR: unbound pwqs
      indexed by node */
};
```

```
  typedef void (*work_func_t) (struct work_struct *work);
    struct work_struct {
      atomic_long_t data;
      struct list_head entry;
      work_func_t func; // обработчик работы
  #ifdef CONFIG_LOCKDEP
      struct lockdep_map lockdep_map;
  #endif
  };
```

```
    extern void raise_softirq(unsigned int nr)
     {
      unsigned long flags;
      local_irq_save(flags);\\ сохраняет состояние флага IF
      \\(разрешает/запрещает прер—я на процессоре)
      raise_softirq_irqoff(nr);
      local_irq_restore(flags);
     }
```

Сокеты

```
#include <net/socket.c>
asmlinkage long sys_socketcall(int call, unsigned long *args)
// ее текст = switch, переключающий ядро на разные функции, связанные с
   сокетом
{
  int err;
  if copy_from_user(a, args, nargs[call])
  return -EFAULT;
  a0 = a[0];
  a1 = a[1];
  switch(call)
  {
    case SYS_SOCKET: err= sys_socket(a0, a1, a[2]); break;
    case SYS_BIND: err= sys_bind(a0, (struct sockaddr*)a1, a[2]); break;
    case SYS_CONNECT: err= sys_connect(...); break;
    ...
    default: err = -EINVAL; break;
  }
  return err;
}
```

```
  asmlinage long sys_socket(int family, int type, int protocol)
  {
    int retval;
    struct socket *sock;
    ...
    retval = sock_create(famaly, type, protocol, &sock);
    ...
    return retval;
  }
```

```
        struct socket // нет в 6 версии ядра
        {
          socket_state state;
          short type;
          unsigned long flags;
          const struct proto_ops *ops;
          struct fasync_strcut *fasync_list;
          struct file *file;
          struct sock *sk;
          wait_queue_head_t wait;
        }
```

```
  struct sockaddr
  {
    sa_family_t sa_family;
    char sa_data[14];
  }
```

```
struct soackaddr_in
{
  sa_family_t sa_family;
  unsigned short int sin_port;
  struct in_addr sin_addr;
  unsigned char sin_zero[sizeof(struct sockaddr) - sizeof(sa_family_t) -
      sizeof(uint16_t) - sizeof(struct in_addr)];
};
```

Proc

Листинг 1..1: Структура proc_dir_entry

```
<linux/proc_fs.h>
struct proc_dir_entry {
 atomic_t in_use;
 refcount_t refcnt;
 struct list_head pde_openers;
 spinlock_t pde_unload_lock; // Собственное средство взаимоисключения
 ...
 const struct inode_operations *proc_iops; // Операции определенные на
    inode фс proc
 union {
  const struct proc_ops *proc_ops;
  const struct file_operations *proc_dir_ops;
 };
 const struct dentry_operations *proc_dops; // Используетя для регистрации
     своих операций над файлом в proc
 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;
         ...
 loff_t size;
 struct proc_dir_entry *parent;
 ...
 char *name;
 u8 flags;
         ...
}
```

Листинг 1..2: Структура proc_ops

```
struct proc_ops {
 unsigned int proc_flags;
 int (*proc_open)(struct inode *, struct file *);
          // в таблице открытых файлов об одном файле находится столько запи
              сей, сколько раз он был открыт
 ssize_t (*proc_read)(struct file *, char __user *, size_t, loff_t *);
 ...
 ssize_t (*proc_write)(struct file *, const char __user *, size_t, loff_t
    *);
 loff_t (*proc_lseek)(struct file *, loff_t, int);
 int (*proc_release)(struct inode *, struct file *);
 ...
 long (*proc_ioctl)(struct file *, unsigned int, unsigned long);
}
```

Листинг 1..3: Функция proc_create_data

```
extern struct proc_dir_entry *proc_create_data(const char *, umode_t,
  struct proc_dir_entry *,
  const struct proc_ops *, void *);
```

Листинг 1..4: Функция proc_create

```
static inline struct proc_dir_entry *proc_create(const char *name, umode_t mode,
        struct proc_dir_entry *parent,
        const struct proc_ops *proc_ops)
{
 return proc_create_data(name, mode, parent, proc_ops, NULL);
}
```

Листинг 1..5: Хз чо это

```
#include <linux/types.h>
#include <linux/fs.h>

extern struct proc_dir_entry *proc_symlink(const char *, struct proc_dir_entry *, const char *);
extern struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *);
struct proc_dir_entry *create_proc_read_entry( const char *name, mode_t mode, struct proc_dir_entry *base, read_proc_t *read_proc, void *data );
```

```
  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 *);
  ...
  int (*open) (struct inode *, struct file *);
  ...
  int (*release) (struct inode *, struct file *);
  ...
} __randomize_layout;
```

```
unsigned long __copy_to_user(void __user *to, const void *from, unsigned long n);
unsigned long __copy_from_user(void *to, const void __user *from, unsigned long n);
```

Super_block

```
struct super_block {
        struct list_head        s_list;
        dev_t                   s_dev;              // устройство, на котором находится ФС
        unsigned long           s_blocksize;        // размер блока в байтах
        unsigned char           s_dirt;             // флаг изменения супрблока
        struct file_system_type s_type;             // в ядре есть структура описывающая тип ФС
        struct super_operations s_op;               // операции на суперблоке
        struct block_device *b_dev // описывает устройство, на котором находится файловая система (соответствует драйверу блочного устройства)
        unsigned long            s_magic;           // магический номер смонтированной ФС
        struct dentry            *s_root;       //  точка монтирования ФС
        ...
        int                      s_count;         // число ссылок
        struct list_head         s_dirty;           // список измененных inode'ов
        char                     s_id[32];        // имя?
};
```

```
<linux/fs.h>

struct super_operations
{
  struct inode *(alloc_inode)(struct superblock *sb);
void (*destroy_inode) (struct inode *);
...
void (*dirty_inode)(struct inode *, int flags);
int (*write_inode)(struct inode*, struct wirteback_cintrol *wbc);
int (*drop_inode)(struct inode *);
...
void (*put_super)(struct super_block *);
}
```

```
static struct super_block *alloc_super (
  struct file_system_type *type,
  int flags,
  struct user_namespace *user_ns
)
{
  struct superblock *s = kzalloc(sizeof(struct super_block), GFP_USER);
  static const struct super_operations default_ops;
  if (!s) return NULL;
  INIT_LIST_HEAD(&s->s_mounts);
  ...
  INIT_LIST_HEAD(&s->s_inodes);
  ...
}
```

Dentry

```
struct dentry_operations
{
  int (*drevalidate)(struct dentry *, unsigned int);
  ...
  int (*d_hash)(const struct dentry *, unsigned int);
  int (* d_compare)(const struct dentry *, unsigned int, const char *,
      const struct
  qstr *);
  int (* d_delete)(const struct dentry *);
  int (* d_init)(const struct dentry *);
  int (* d_release) (struct dentry *);
  void (* d_input)(struct dentry *, struct inode *);
  char *(* d_name)(struct dentry *, char *, int);
  ..
}
```

Inode

```
struct inode {
  struct list_head i_hash;
 struct list_head i_list;
 struct list_head i_dentry;
 ...
 unsigned long i_ino;
 atomic_t i_count;
 kdev_t i_rdev;
 umode_t i_mode;
 ...
 loff_t i_size;
 ...
 // информация о времени модификации и доступа к inode
 ...
 // 6 полей, связ с блоками(только для ядра)
 ...
 unsigned int i_blkbits;// битовая карта блока
 unsigned long i_blksize;// размер блоков
 unsigned long i_blocks;// кол-во блоков
 ...
 const struct inode_operations  *i_op; //перечень функций определенных для
      работы с inode и с открытыми файлами
 const struct file_operations  *i_fop;
 struct super_block  *i_sb;
 ...
 struct list_head i_devices;
 struct pipe_inode_info *i_pipe;
 struct block_device *i_bdev;
 struct char_device *i_cdev;
 ...
 unsigned long i_state;
 unsigned int i_flags;
 ...
 union //типы фс
 {
 struct minix_inode_info minix_i;
 struct ext2_inode_info ext2_i;
 ....
 struct ntfs_inode_info ntfs_i;
 struct msdos_inode_info msdos_i;
 ...
 struct nfs_inode_info nfs_i;// сетевая фс
 struct ufs_inode_info ufs_i;
 ...
 struct proc_inode_info proc_i;
 struct socket socket_i;
 ...

 }
};
```

```
    struct inode_operations {
  struct dentry * (*lookup) (struct inode *,struct dentry *, unsigned int)
     ;

  int (*create) (struct inode *,struct dentry *,
           umode_t, bool);

  int (*mkdir) (struct inode *,struct dentry *,
           umode_t);

  int (*rename) ( struct inode *, struct dentry *,
      struct inode *, struct dentry *, unsigned int);
  ...
} ;
```

file

```
    struct file {
  struct path     f_path;
  struct inode     *f_inode;  /* cached value */
  const struct file_operations  *f_op;
          ...
  atomic_long_t     f_count;// кол-во жестких ссылок
  unsigned int       f_flags;
  fmode_t          f_mode;
  struct mutex      f_pos_lock;
  loff_t          f_pos;
  ...
  struct address_space  *f_mapping;
  ...
};
```

```
  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 *);
  ...
  int (*open) (struct inode *, struct file *);
  ...
  int (*release) (struct inode *, struct file *);
  ...
} __randomize_layout;
```

mount

```
typedef int (*fill_super_t)(struct super_block *, void *, int);
struct dentry *mount_bdev(struct file_system_type *fs_type, int flags,
    const char *dev_name, void *data, fill_super_t fill_super);
```

```
struct dentry *mount_nodev(struct file_system_type *fs_type, int flags,
    void *data, fill_super_t fill_super);
struct dentry *mount_single(struct file_system_type *fs_type, int flags,
    void *data, fill_super_t fill_super);
```

Simple & generic

```
int generic_delete_inode(struct inode *inode)
{
  return 1;
}
```

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

file_system_type

```
struct file_system_type {
  const char *name;
  int fs_flags;
  #define FS_REQUIRES_DEV     1
  ...
  #define FS_USERNS_MOUNT     8   /* Can be mounted by userns root */
  ...
  struct dentry *(*mount) (struct file_system_type *, int,
  const char *, void *);
  void (*kill_sb) (struct super_block *);
  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 module *owner;
    ...
  };
```

Seqfile

Листинг 1..6: Структуры

```
struct seq_operations;

struct seq_file {
 char *buf;
 size_t size;
 size_t from;
 size_t count;
 ...
 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);
};
```