int alloc_workqueuedevice_driver

```
struct device_driver {
      const char *name;
      struct bus_type *bus;
      struct module *owner;
      const char mod_name;    //имя модуля (для
встраиваемых модулей)
      ...
      //точки входа в драйвер
      //вызывается для запроса существования
конкретного устройства, если драйвер может
с ним работать
      int (*probe)(struct device *dev);
      //удаляет драйвер
      int (*remove)(struct device *dev);
      //отключает драйвер
      void (*shutdown)(struct device *dev);
      //переводит драйвер в режим сна
      int (*resume)(struct device *dev);
      //выводит драйвер из сна
      int (*suspend)(struct device *dev);
      ...
}
```

usb_driver

```
struct usb_driver {
     const char * name;
     ...
     struct file_operations *fops;
     mode_t mode;
     int minor_base;
     ...
     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);
     ...
     struct device_driver driver;
     const struct usb_device_id * id_table;
};
```

usb_device

https://vk.com/photo-142592581_457274108

```
struct usb_device {
    int devnum;
    char devpath[16];
    enum usb_device_state state;
    ...
    struct usb_host_endpoint ep0;
    struct device dev;
    struct usb_device_descriptor descriptor;
    ...
    char * product;
    char * manufacturer;
    char * serial;
    ...
}
```

device

```
struct device {
     struct device *parent;  //устройство, к которому
подключается новое устройство. Обычно шина
или хост-контроллер
     const char *init_name;  //первоначальное имя
устройства
     const struct device_type *type;
     struct mutex mutex;     //для синхронизации
вызовов устройств
     struct bus_type *bus;   //тип шины, к которой
подключено устройство
     struct device_driver *driver;
     ...
     #ifdef CONFIG_GENERIC_MSI_DOMAIN
     struct irq_domain *msi_domain;
     #endif
     ...
     //DMA
     ...
     struct device_dma_parameters *dma_params; //структура
низкого уровня
     ...
}
```

tasklet_struct

```
struct tasklet_struct
 {
   struct tasklet_struct *next;  /* указатель на следующий тасклет в     списке */
   unsigned long state;       /* состояние тасклета */
   atomic_t count;            /* счетчик ссылок */
   void (*func) (unsigned long); /* функция-обработчик тасклета*/
   unsigned long data; /* аргумент функции-обработчика тасклета */
);
```

work_struct

```
struct work_struct
{
    atomic_longg_t data;
    struct list_head entry;
    work_funt_t func;
    #ifdef CONFIG_LOCKED struct lockder_map;
    #endif
};
```

workqueue_struct

```
struct workqueue_struct // определена в <linux/workqueue.h>
{
    unsigned int flags;
    union{
            struct epu_workqueue_struct_perepu *pepu;
            struct epu_workqueue_struct *single;
            unsiggned long v;
    } epu wq;
    struct list_head list; // список всех очередей работ

    struct mutex flush_mutex; // защищает wq flushing

    int work_color;
    int flush_color;
    atomic_t nr_cwqs_to_flush;
    struct wq_flusher *first_flushes;
    struct list_head flusher_queue;
    struct list_head flusher_overflow;
    mayday_mask_t mayday_mask;
    struct worker *rescuer;
    int nr_drainers;
    int saved_max_active; // связано с количеством работ

    ...
    char name[];
}
```

cpu_workqueue_struct

```
struct cpu_workqueue_struct
{
    spinlock_t lock;
    long remove_sequence;
    long insert_sequence;
    struct list_heaad worklist;
    wait_queue_head_t more_work;
    wait_queue_head_t done;
    struct workqueue_struct *wq;
    task_t *thread;
    int run_depth;
};
```

proc_dir_entry

```
struct proc_dir_entry {
        unsigned short low_ino;     //номера inode файла
        unsigned short namelen;
        const char *name;           //имя виртуального
файла
        mode_t mode;                //права доступа
        nlink_t nlink;
        uid_t uid;
        gid_t gid;
        unsigned long size;
        struct inode_operations * ops;
        //функции чтения и записи
        int (*get_info)(char *buffer, char **start,
                off_t offset, int length, int unused);
        void (*fill_inode)(struct inode *);
        struct proc_dir_entry *next, *parent, *subdir;
        void *data;
};
```

struct task_struct

```
struct task_struct
{
    struct task_struct *next_task, *prev_task;
    ...
    char comm[16]; // имя файла
    ...
    int pid; // идентификатор процесса
    int parent; // id предка
    ...
}
```

struct files_struct

```
struct files_struct
{
    atomic_t count; // счетчик кол-ва такой структуры
    spinlock_t file_block;
    ...
    int next_fd; // кол-во дескрипторов файлов
    struct file **fd; // массив всех файловых объектов
    ...
    struct file * fd_array[NR_OPEN_DEFAULT]; // массив файлов открытых процессом
};
```

struct fs_struct

```
struct fs_struct
{
    atomic_t count;
    rwlock_t lock;
    ...
    int umask; // маска создания файла
    struct dentry *root; // корневая директория
    struct dentry *pwd; // текущая директория
    ...
    struct vfsmount *rootmnt;
};
```

struct vfsmount

```
struct vfsmount {
      struct dentry *mnt_root;
      struct super_block *mnt_sb;
      int mnt_flags;
}
```

struct file

```
struct file
{
  struct path f_path;
  struct inode *f_node; // cached value
  const struct file_operations *f_op; // файловые операции
  spinlock_t f_lock; // своя спинблокировка (средство взаимоисключения);
  atomic_long_t f_count; // счётчик ссылок
  unsigned int f_flags;  // устанавливаются СВ open()
  fmode_t f_mode;       // режим доступа к файлу
  loff_t f_pos;          // смещение в логическом файле, на это поле
};
```

struct super_block

```
struct super_block
{
   struct list_head s_list;
   kdev_t s_dev;                    // устройство, на котором находится ФС
   unsigned long s_blocksize;       // размер блока в байтах
   unsigned char s_dirt;                  // флаг изменения
  unsigned long s_max_byte;               // максимальный размер файла
  struct file_system_type *s_type; // описывает тип ФС (всегда один)
  struct super_operations *s_op;   // операции, определённые на суперблоке
  ...
  unsigned long s_magic;           // магический номер ФС
  struct dentry *s_root;           // точка монтирования ФС, или каталог монтирования ФС
  ...
  int s_count;                           // счетчик ссылок на суперблок
  ...
  struct list_head s_dirty;   // список измененных индексов
  ...
  struct block_device *s_bdev; // драйвер соответствующего блочного устройства
  ...
  char s_id[32];                         // строка имени
  ...
}
```

struct file_operations

```
struct file_operations
{
  struct module *owner;
  loff_t (*llseek)(struct file*, loff_t, int); // установить файловый указатель
  ssize_t (*read)(struct file*, char _user *, size_t, loff_t *); // чтение
  ssize_t (*write)(struct file *, const char _user*, size_t, loff_t*); // запись
  …
  int (*open)(struct inode*, struct file *); // открытие
  int (*flush)(struct file*, fl_owner_t id); // запись содержимого буфера в файл
  int (*release)(struct inode*, struct file*);// закрытие
}
```

struct inode

```
struct inode
{
  umode_t i_mode;                                  // права доступа
  unsigned short i_opflags;
  atomic_t i_count;                          // счетчик ссылок
  unsigned int i_nlink;                 // количество жестких ссылок
  kuid_t i_uid;                              // идентификатор пользователя-владельца
  kgid_t i_gid;                              // идентификатор группы-владельца
  unsigned int i_flags;
  ...
  const struct inode_operations *i_op; // указатель на структуру inode_operations
  struct super_block *i_sb;                        // связанный суперблок
  struct address_space *i_mapping;          // связанное отображение
  ...
  unsigned long i_ino;                 // номер индекса
  ...
  loff_t i_size;                             // размер файла в байтах
  struct timespec i_atime;             // время последнего доступа к файлу
  struct timespec i_mtime;             // время последнего изменения файла
  struct timespec i_ctime;            // время изменения индекса
  ...
  struct hlist_node i_hash;            // хешированный список
  struct list_head i_dentry;           // список объектов dentry
  …
  const struct file_operations *i_op;
}
```

struct nameidata

```
struct nameidata
{
    struct dentry *dentry;
    struct vfsmount *mnt;
    struct qstr last;
}
```

struct dentry

```
{
  // RCU lookup touched fields
  unsigned int d_flags;       // protected by d_lock
  seqount_t d_seq;                    // per detry seqlock
  struct dentry *d_parent;
  struct qstr d_name;         // dehntry name
  struct inode *d_inode;    // связанный inode
  unsigned char d_iname[DNAME_INLINE_LEN]; // короткое имя
  ...
  const struct dentry_operations *d_op;
  struct super_block *d_sb;
  unsigned long d_time;
  void *d_fsdata;
  struct list_gead d_lru;
  ...
  struct list_head d_subdirs;
}
```