嵌入式工程师自我修养

Linux内核编程：文件系统

主讲：王利涛

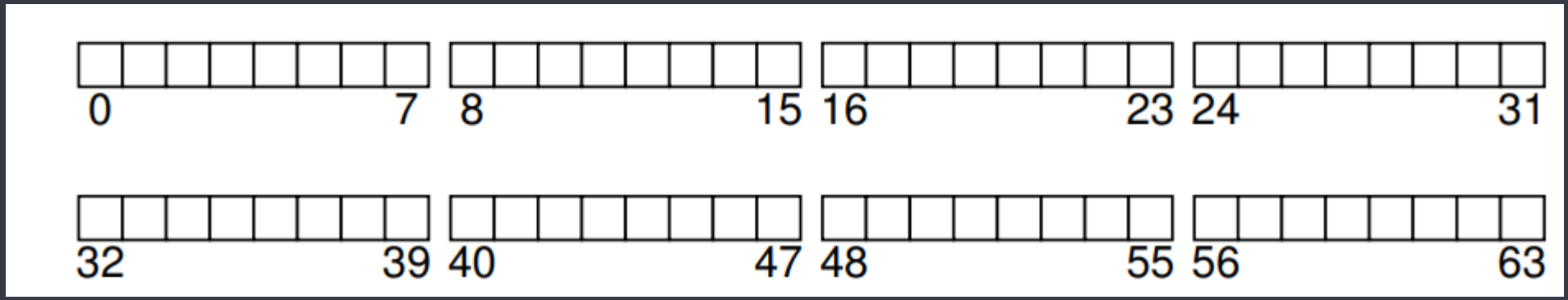
- 文件系统的基本概念
  - 分区、格式化、挂载、卸载
  - 文件系统类型：EXT、FAT、YAFFS
  - super_block、inode、dentry、file、
  - 虚拟文件系统：VFS
  - 设备文件、设备节点
  - 根文件系统、根目录、rootfs
  - Ramdisk、initrd、initramfs、rdinit
  - Busybox、init、linuxrc
  - bootcmd、bootargs …
  - …

- 本期课程的预期收获
  - 基本概念：分区、格式化、挂载、卸载
  - 文件系统框架
    - 主机端：软件架构
    - 设备端：数据存储
  - 虚拟文件系统：VFS
  - 磁盘、设备文件的IO操作流程
    - 格式化、挂载
    - 创建文件、打开文件
    - 文件的读写
  - 根文件系统：
    - NFS、磁盘
    - ramdisk/initrd
    - initramfs
  - ramfs、tmpfs

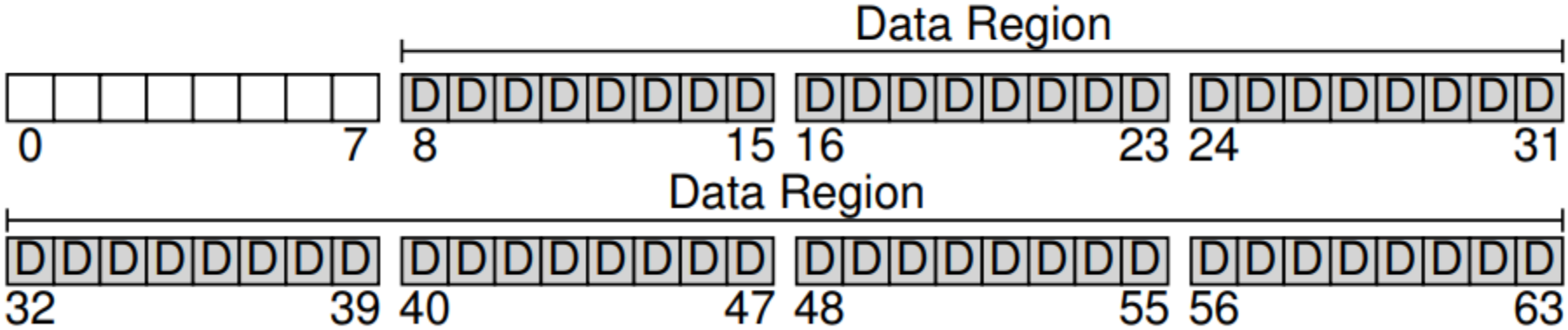
- 实验平台
  - 虚拟开发板： vexpress A9 ARM board
  - U-boot + linux-5.10.x + NFS 开发环境
  - Vmware station + Ubuntu-20.04/16.04
  - 获取镜像
    - 关注公众号：宅学部落
    - 输入：小宅实验室
    - 文档指南：www.zhaixue.cc/qemu教程

01 什么是文件系统？
设备端

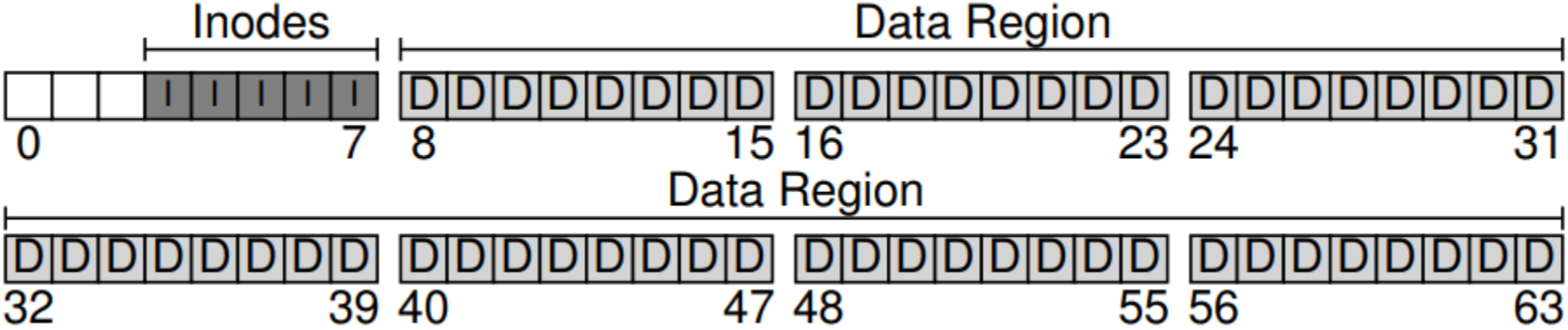
- 数据的存储
  - 磁盘：磁道、扇区
  - NAND Flash： Page、 Block
    - Linux系统编程01期：揭开文件系统神秘的面纱.pdf
    - operating systems three easy pieces



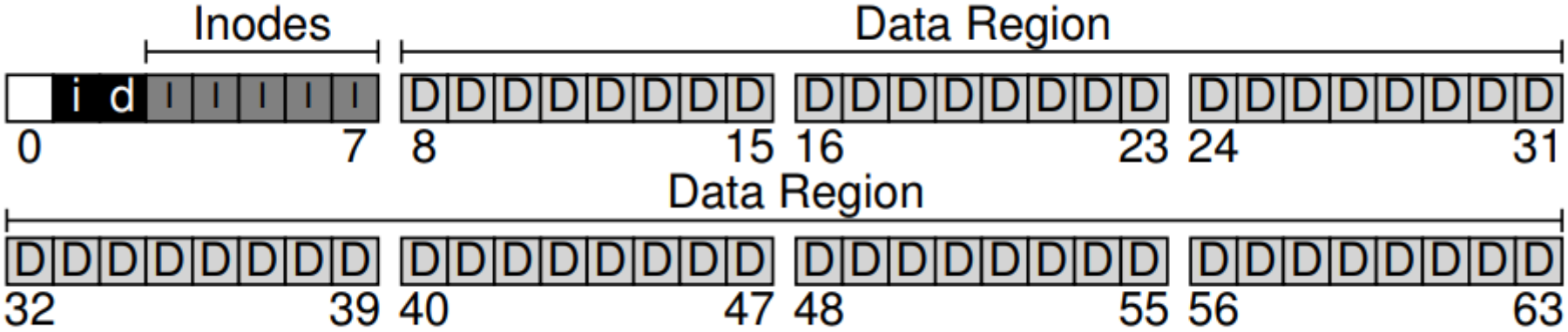
- 纯数据区和元数据区



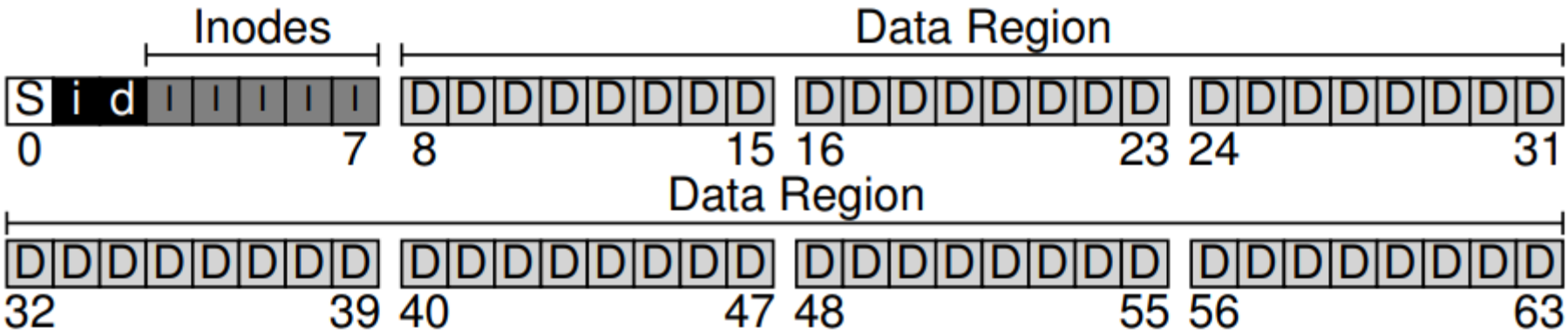
- inode table


Inodes
Data Region
I I I I I
D D D D D D D D
D D D D D D D D
D D D D D D D D
0
7
8
15
16
23
24
31
Data Region
D D D D D D D D
D D D D D D D D
D D D D D D D D
D D D D D D D D
32
39
40
47
48
55
56
63

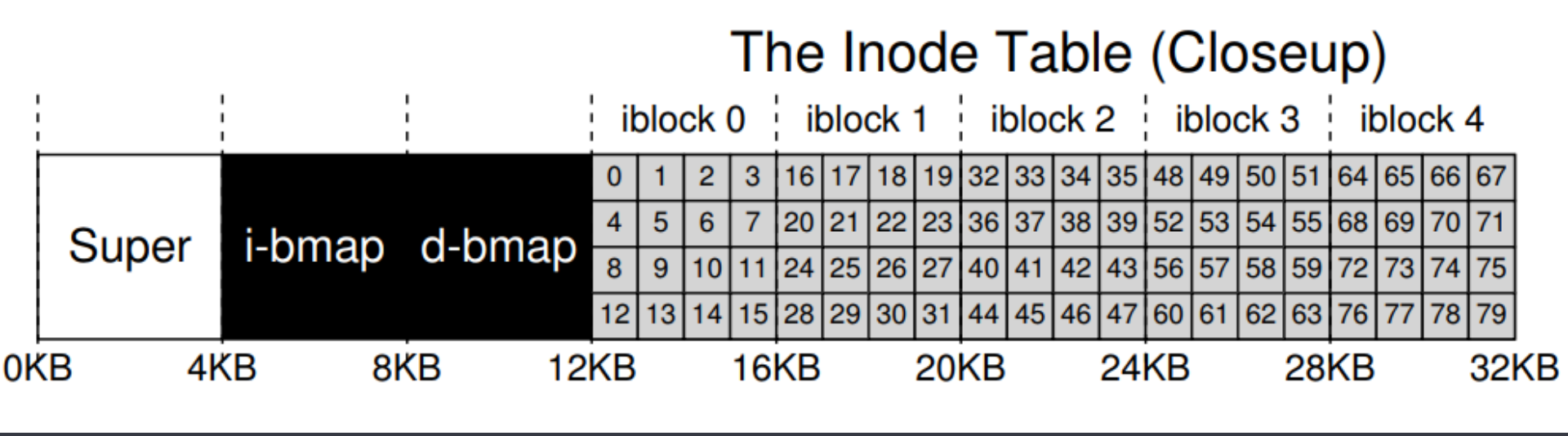
- data bitmap and inode bitmap



- superblock


Inodes
Data Region
S i d I I I I I
D D D D D D D D
D D D D D D D D
D D D D D D D D
0 7 8 15 16 23 24 31
Data Region
D D D D D D D D
D D D D D D D D
D D D D D D D D
D D D D D D D D
32 39 40 47 48 55 56 63

- 小结


The Inode Table (Closeup)
iblock 0
iblock 1
iblock 2
iblock 3
iblock 4
Super
i-bmap
d-bmap
0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 64 65 66 67
4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 68 69 70 71
8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 72 73 74 75
12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63 76 77 78 79
0KB
4KB
8KB
12KB
16KB
20KB
24KB
28KB
32KB

- MINIX文件系统

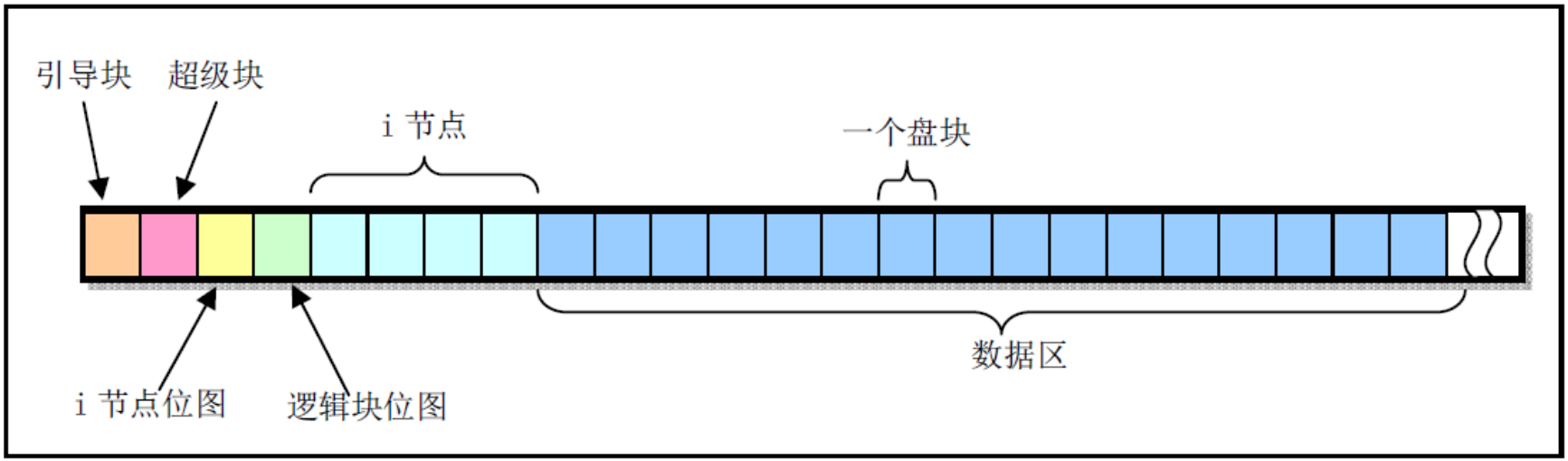
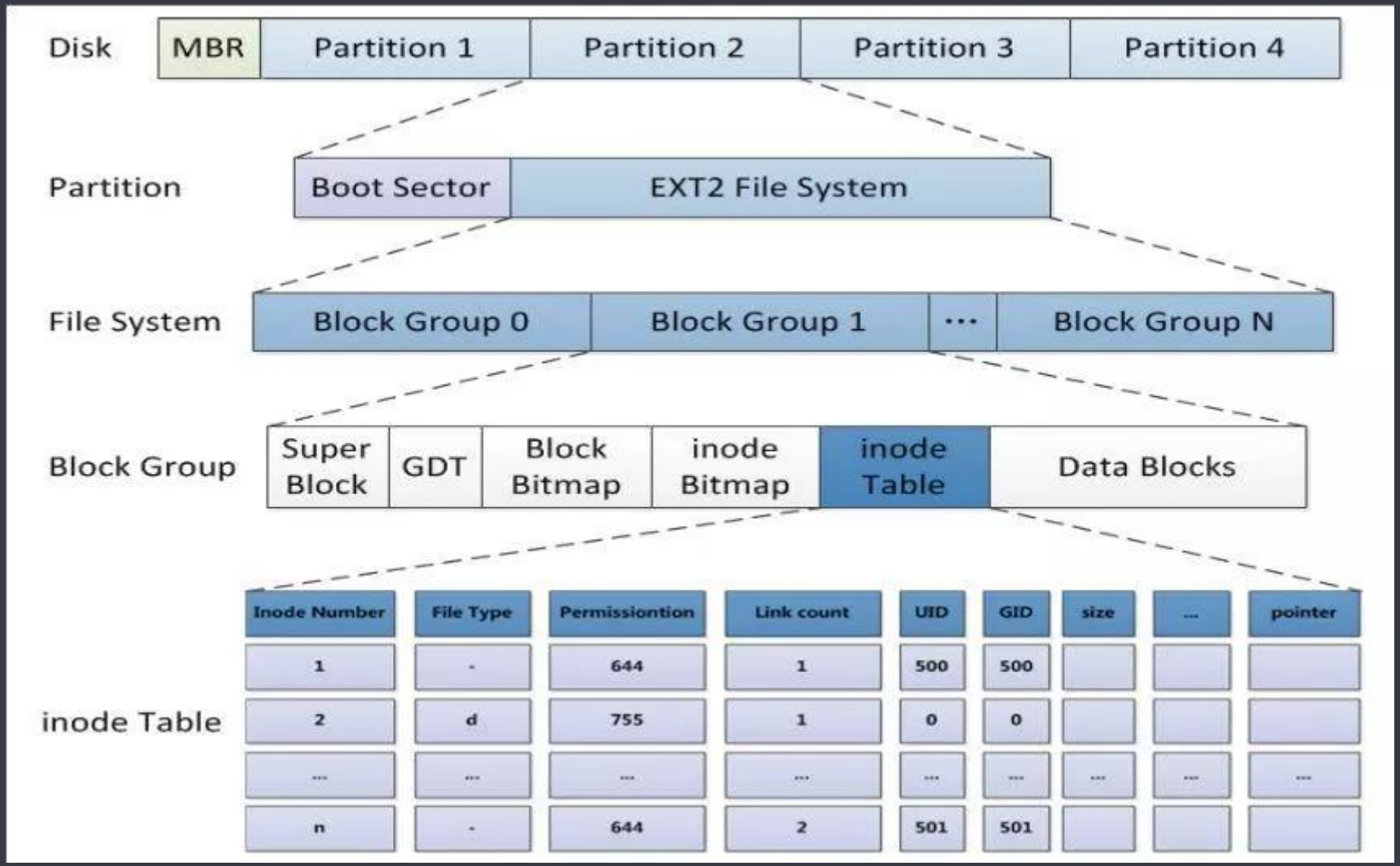


图 12-1 建有 MINIX 文件系统的一个 360K 软盘中文件系统各部分的布局示意图

• EXT2文件系统


Disk
MBR
Partition 1
Partition 2
Partition 3
Partition 4
Partition
Boot Sector
EXT2 File System
File System
Block Group 0
Block Group 1
...
Block Group N
Block Group
Super Block
GDT
Block Bitmap
inode Bitmap
inode Table
Data Blocks
inode Table
Inode Number
File Type
Permissiontion
Link count
UID
GID
size
...
pointer
1 - 644 1 500 500
2 d 755 1 0 0
n - 644 2 501 501

02 磁盘的格式化与挂载

- 实验
  - 在虚拟机上添加一块磁盘
  - 格式化、将数据dump出来
  - 挂载、读写数据
  - 将读写数据dump出来

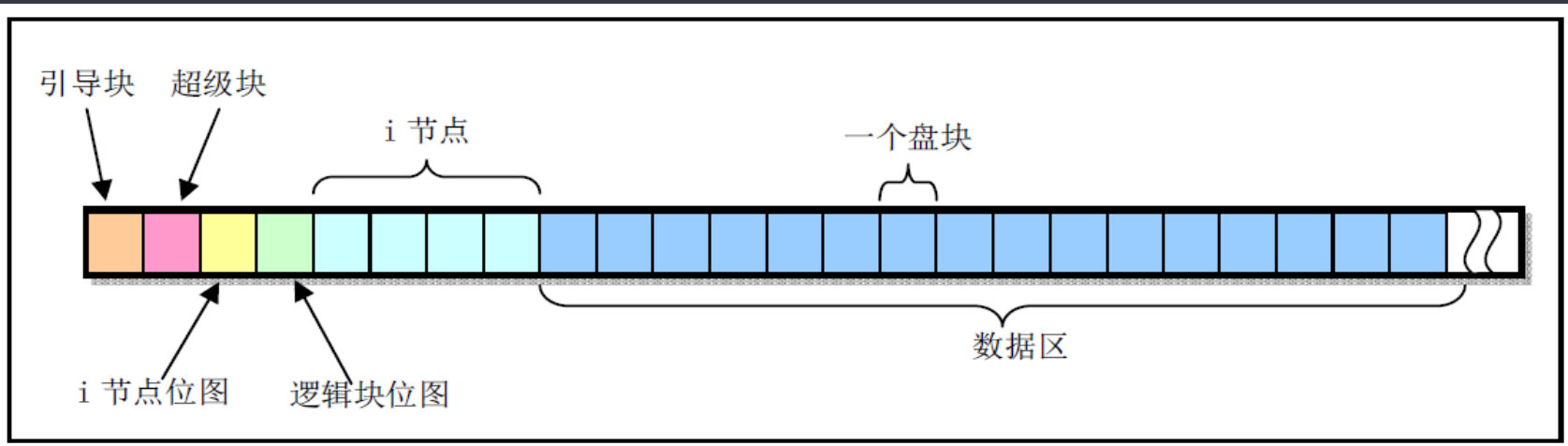
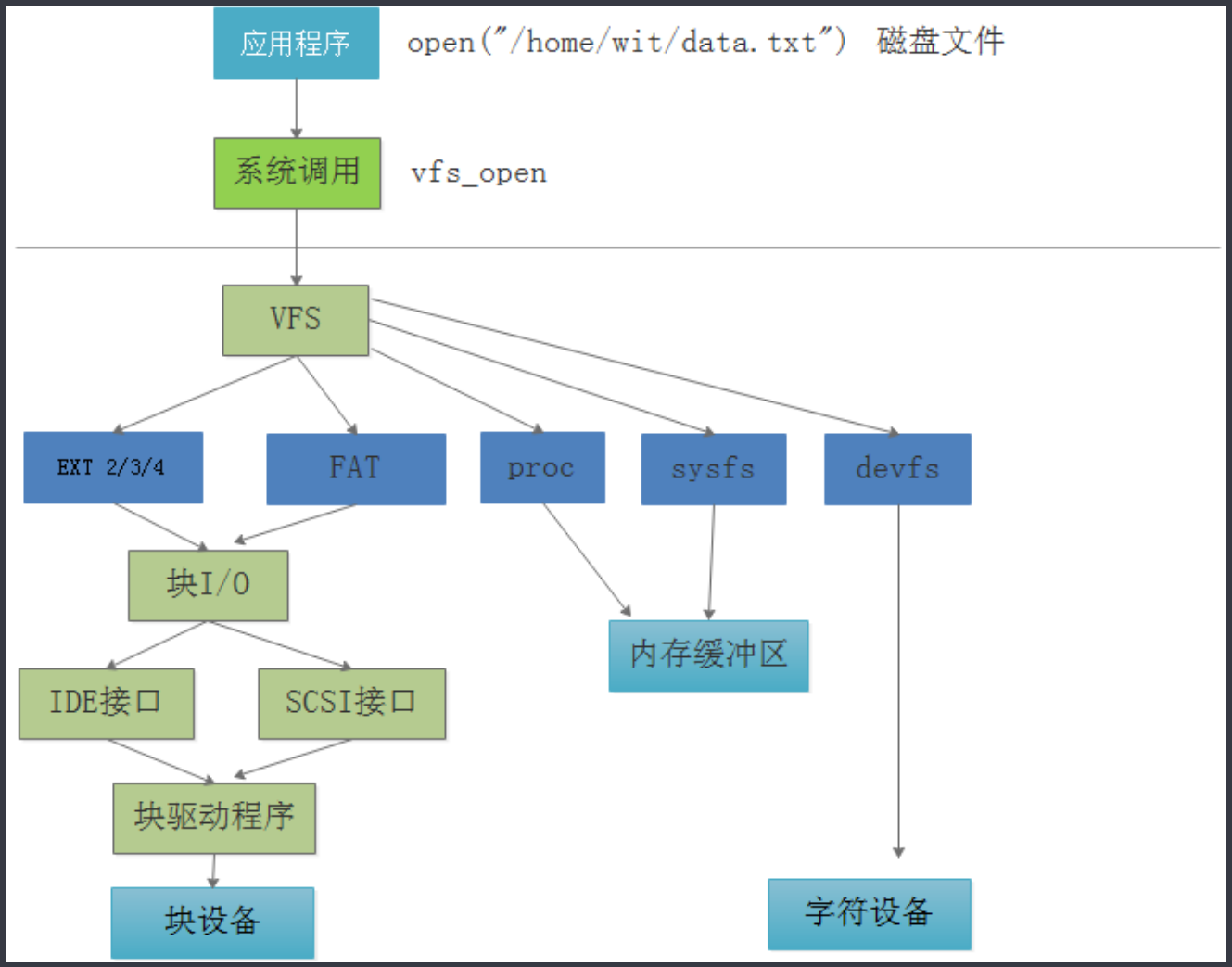


图 12-1 建有 MINIX 文件系统的一个 360K 软盘中文件系统各部分的布局示意图

03 什么是文件系统？
主机端

• 内核中的文件系统


应用程序
open("/home/wit/data.txt") 磁盘文件
系统调用
vfs_open
VFS
EXT 2/3/4
FAT
proc
sysfs
devfs
块I/O
内存缓冲区
IDE接口
SCSI接口
块驱动程序
块设备
字符设备

- File Model
  - file_system_type
  - super_operations
  - inode_operaitons
  - detry_operations
  - file_operations

ext2
super.c：
    file_system_type
    super_operations
namei.c：
    inode_operations
file.c：
    file_operations xxx_file_fops
dir.c:
    file_operations xxx_dir_fops

9p
vfs_super.c：
  super_operations
vfs_inode.c：
  inode_operations
vfs_denrty.c：
  dentry_operations
vfs_file.c：
vfs_dir.c：
  file_operations
vfs_addr.c：
  address_space_operations

debugfs:
inode.c：
  inode_operaitons degbugfs_file_inode_operations
  inode_operaitons degbugfs_dir_inode_operations
  inode_operaitons degbugfs_symlink_inode_operations
  super_operations debugfs_super_operations
  detry_operations debugfs_dops
  file_system_type debug_fs_type
file.c：
  file_operations xxx_file_fops
  file_operations xxx_dir_fops

04 文件系统核心数据结构：super_block

• MINIX的超级块

```
/*  minix super-block data on disk */
struct minix_super_block {
        __u16 s_ninodes;
        __u16 s_nzones;
        __u16 s_imap_blocks;
        __u16 s_zmap_blocks;
        __u16 s_firstdatazone;
        __u16 s_log_zone_size;
        __u32 s_max_size;
        __u16 s_magic;
        __u16 s_state;
        __u32 s_zones;
};
```

```
/* V3 minix super-block data on disk */
struct minix3_super_block {
        __u32 s_ninodes;
        __u16 s_pad0;
        __u16 s_imap_blocks;
        __u16 s_zmap_blocks;
        __u16 s_firstdatazone;
        __u16 s_log_zone_size;
        __u16 s_pad1;
        __u32 s_max_size;
        __u32 s_zones;
        __u16 s_magic;
        __u16 s_pad2;
        __u16 s_blocksize;
        __u8  s_disk_version;
};
```

05 文件系统核心数据结构：inode

- inode

```
struct minix_inode {
        __u16 i_mode;
        __u16 i_uid;
        __u32 i_size;
        __u32 i_time;
        __u8  i_gid;
        __u8  i_nlinks;
        __u16 i_zone[9];
};

struct minix2_inode {
        __u16 i_mode;
        __u16 i_nlinks;
        __u16 i_uid;
        __u16 i_gid;
        __u32 i_size;
        __u32 i_atime;
        __u32 i_mtime;
        __u32 i_ctime;
        __u32 i_zone[10];
};
```

- inode_operations

```
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 inode *, int);
        struct posix_acl * (*get_acl)(struct inode *, int);

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

        int (*create) (struct inode *,struct dentry *, umode_t, bool);
        int (*link) (struct dentry *,struct inode *,struct dentry *);
        int (*unlink) (struct inode *,struct dentry *);
        int (*symlink) (struct inode *,struct dentry *,const char *);
        int (*mkdir) (struct inode *,struct dentry *,umode_t);
        int (*rmdir) (struct inode *,struct dentry *);
        int (*mknod) (struct inode *,struct dentry *,umode_t,dev_t);
        int (*rename) (struct inode *, struct dentry *,
                        struct inode *, struct dentry *, unsigned int);
        int (*setattr) (struct dentry *, struct iattr *);
        int (*getattr) (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 *, struct timespec64 *, int);
        int (*atomic_open)(struct inode *, struct dentry *,
                           struct file *, unsigned open_flag,
                           umode_t create_mode);
        int (*tmpfile) (struct inode *, struct dentry *, umode_t);
        int (*set_acl)(struct inode *, struct posix_acl *, int);
} ____cacheline_aligned;
```

06 文件系统核心数据结构：dentry

- dentry

```
struct minix_dir_entry {
        __u16 inode;
        char name[0];
};

struct minix3_dir_entry {
        __u32 inode;
        char name[0];
};
```

- dentry_operations

```
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;
```

07 文件系统核心数据结构：file

- file

```
struct file {
        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;
        spinlock_t              f_lock;
        enum rw_hint            f_write_hint;
        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;
        void                    *private_data;
        struct address_space    *f_mapping;
        errseq_t                f_wb_err;
        errseq_t                f_sb_err; /* for syncfs */
} __randomize_layout    __attribute__((aligned(4)));
```

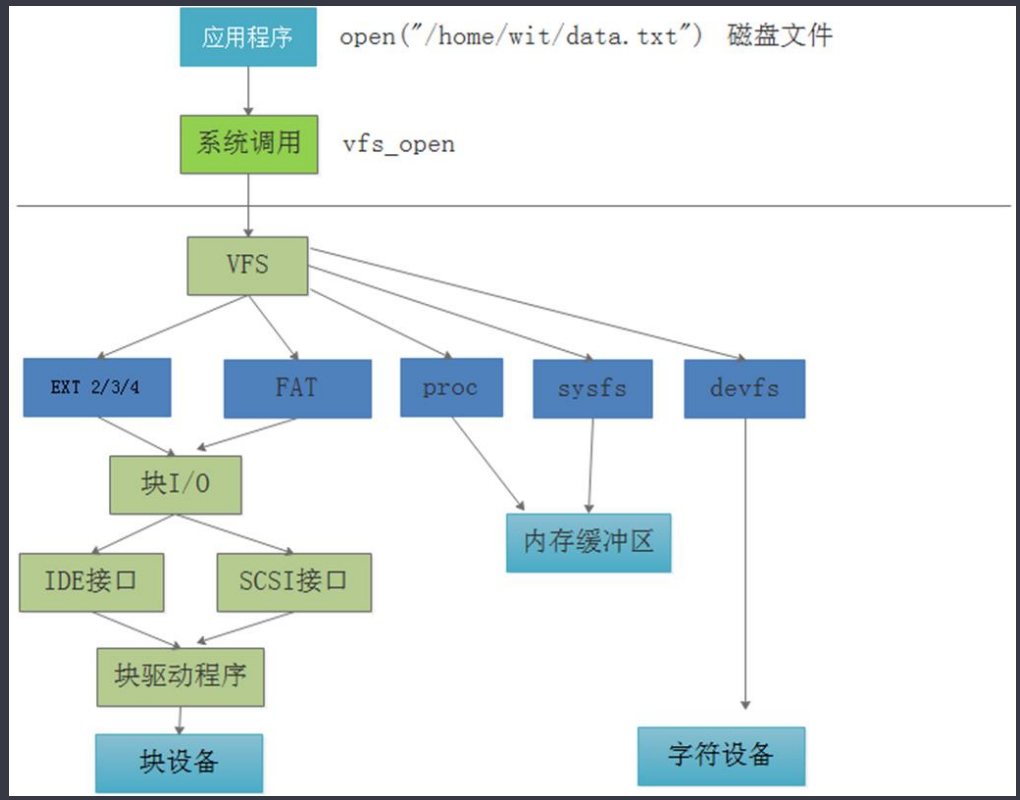
- file_operations

```
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, bool spin);
        int (*iterate) (struct file *, struct dir_context *);
        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 *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        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 *); loff_t len);
        int (*fadvise)(struct file *, loff_t, loff_t, int);
} __randomize_layout;
```

08 虚拟文件系统：VFS

- 虚拟文件系统的作用
  - VFS：文件系统抽象层
    - The Virtual Filesystem
    - Virtual Filesystem Switch
  - 连接操作系统内核和具体文件系统的桥梁
  - 向下的接口：与具体文件系统交互，实现回调
  - 向上的接口：抽象统一系统调用接口
  - 缓存数据：超级块信息、链表、inode…
  - 初始化、挂载、路径名查找…

- VFS 核心数据结构
  - VFS super_block
  - VFS inode
  - VFS dentry
  - VFS file
  - file_system_type
  - vfsmount
  - 全局数组、变量



09 文件系统的注册

• 核心数据结构

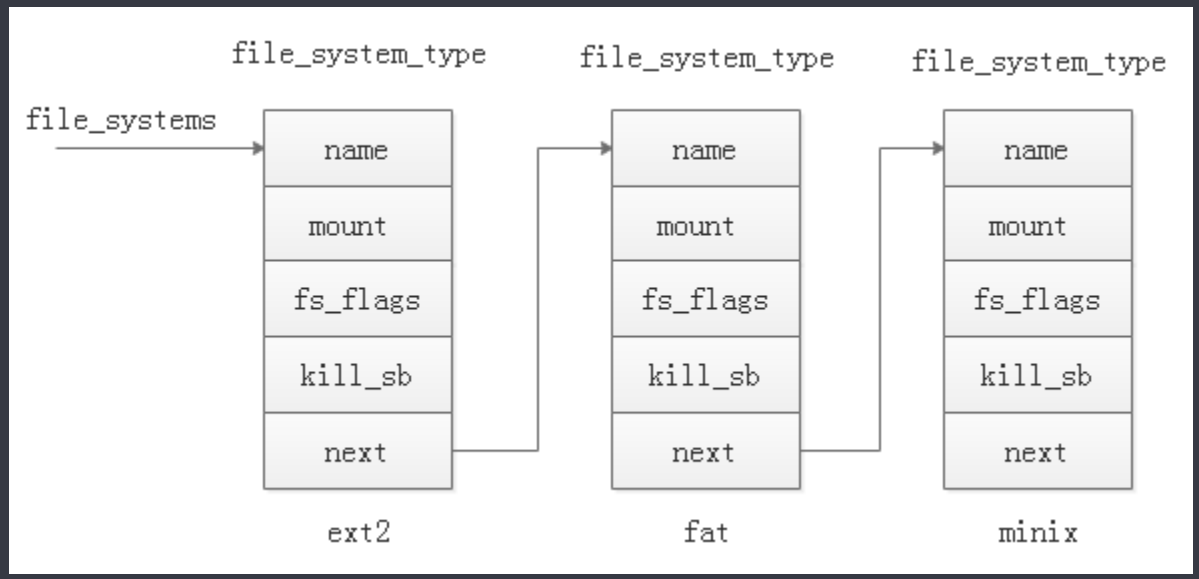
```
struct file_system_type  {
        const char *name;  //每个文件系统类型的名字，必须唯一，不能重复，minix、ext2
        int fs_flags;
        int (*init_fs_context)(struct fs_context *);
        const struct fs_parameter_spec *parameters;
        struct dentry *(*mount) (struct file_system_type *, int, const char *, void *); // mount时回调该函数
        void (*kill_sb) (struct super_block *); //get_sb
        struct module *owner;
        struct file_system_type * next; // 所有挂载实例在内核中构成一个链表
        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 i_mutex_dir_key;
};
```

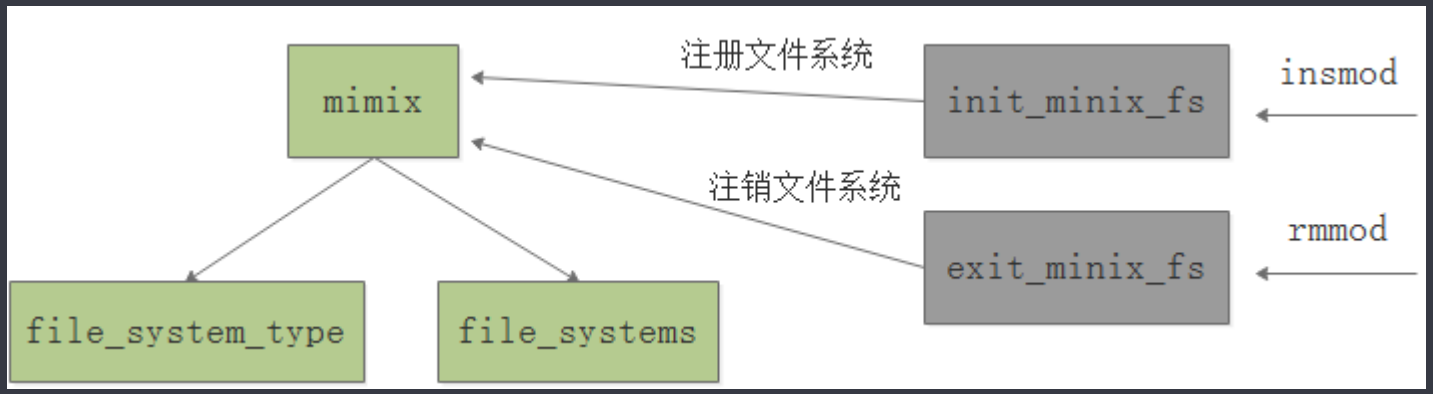
- 文件系统的注册
  - 链表：file_systems
  - 函数：register_filesystem

```
#include <linux/fs.h>
extern int register_filesystem(struct file_system_type *);
extern int unregister_filesystem(struct file_system_type *);
```



• 注册/注销内核源码分析

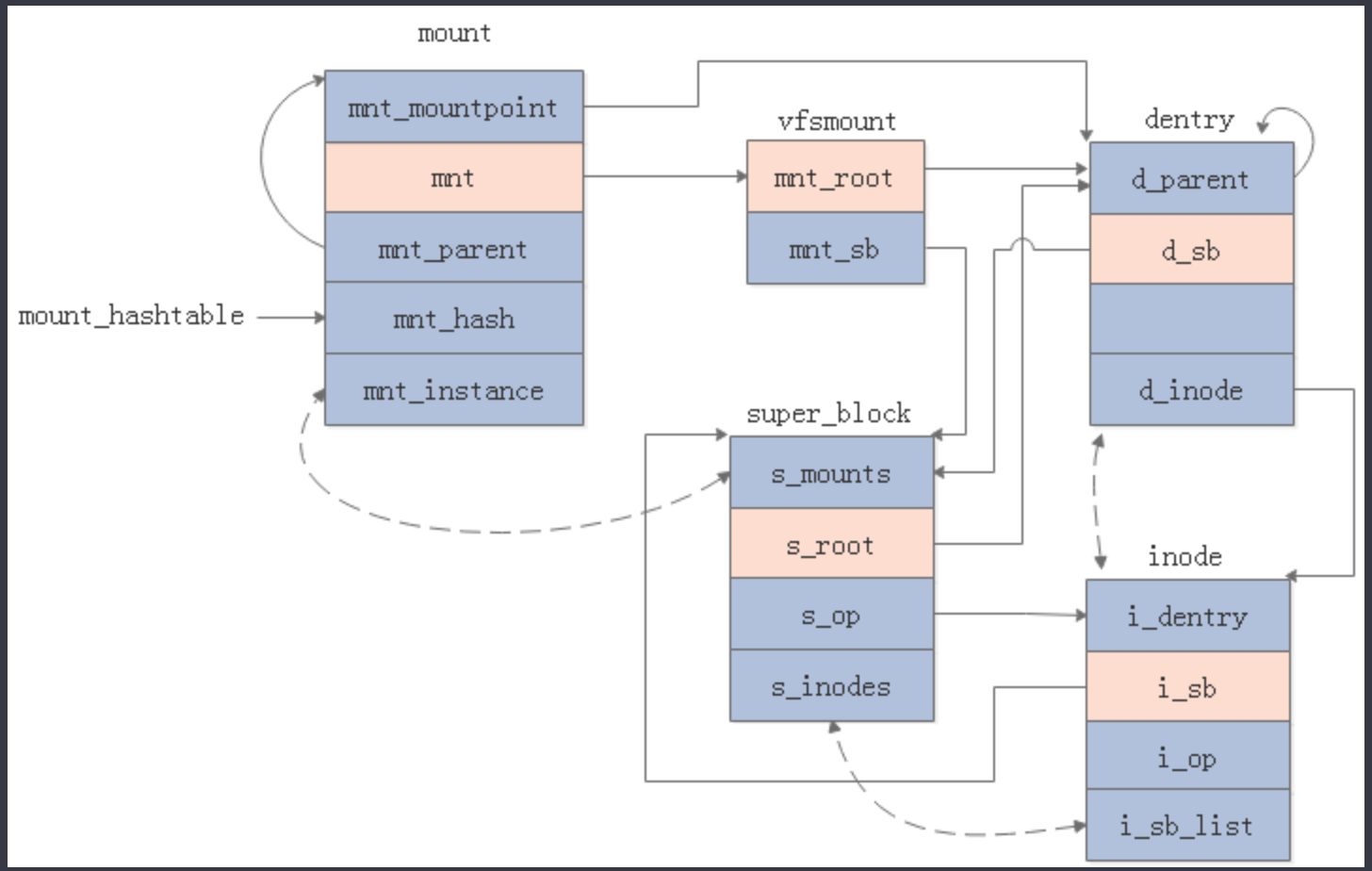
Linux-5.10.4：
Fs/minix/inode.c
init_minix_fs
exit_minix_fs
fs/file_systems.c：
register_filesystem
unregister_filesystem



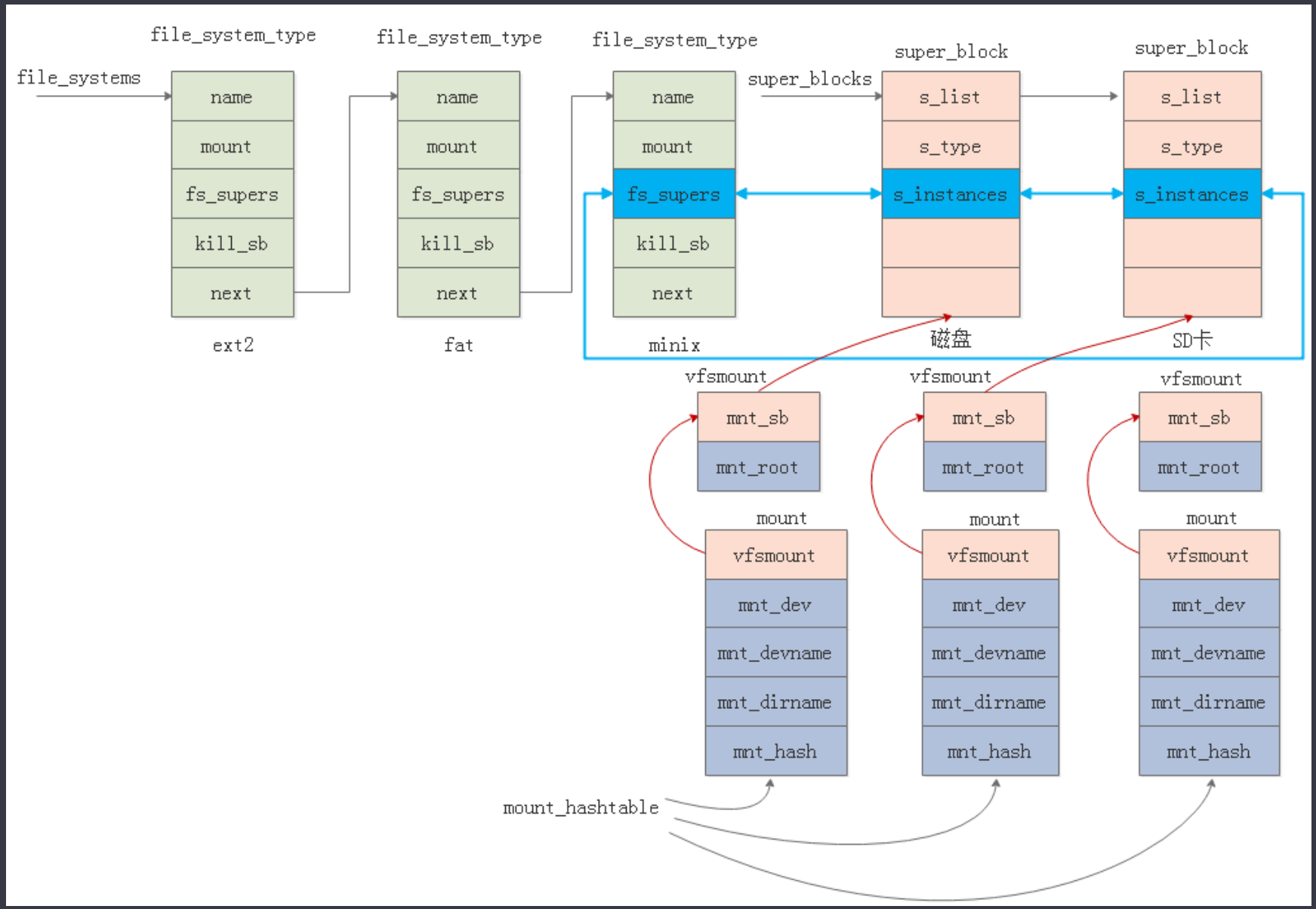
10 文件系统的挂载(上)

• 核心数据结构

- mount、vfsmount、dentry、inode、super_block



• 几个重要的全局变量


file_system_type
file_system_type
file_system_type
super_block
super_block
file_systems
super_blocks
name
mount
fs_supers
kill_sb
next
s_list
s_type
s_instances
ext2
fat
minix
磁盘
SD卡
vfsmount
mnt_sb
mnt_root
mount
vfsmount
mnt_dev
mnt_devname
mnt_dirname
mnt_hash
mount_hashtable

- 挂载后的文件系统全景图
  - 父文件系统：核心数据结构体之间的关联
  - 子文件系统：核心数据结构体之间的关联
  - 父、子文件系统的关联
  - 路径名的解析
- 思考
  - 多个文件挂载同一个目录，底层操作是怎样的？
  - 为什么多次挂载后，只显示最后挂载的内容？

11 文件系统的挂载(下)

• 挂载过程分析

mout /dev/sdc /mnt

fs/namespace.c：

mount

do_mount

get_fs_type

do_add_mount

graft_tree

graft_tree→attach_recursive_mnt→propagate_mnt： 将mount添加到全局数组mnt_hash中

graft_tree→attach_recursive_mnt→mnt_set_mountpoint： 将子mount和父mount建立关联

graft_tree→attach_recursive_mnt→commit_tree→__attach_mnt：将挂载的dentry和vfsmount生成hash数组保存mount_hashtable

12 文件打开过程分析(上)

- 学习重点
  - open(“/mnt/zhaixue.c”)
  - 内核中的核心数据结构及其关联
  - 用户空间路径名到内核空间路径的转换
  - pathname -- path <vfsmount, dentry>
  - 路径解析、有挂载点的路径解析
  - 文件描述符fd和file对象的关联
  - file和 inode、file_operations 的关联
  - 文件I/O回调函数是怎样被调用的？
    - 普通文件
    - 设备文件：字符设备、块设备
    - 管道文件
    - 链接文件

- 路径的转换
  - 用户空间的路径：由/链接的一个字符串
  - 内核空间：超级块、inode、dentry、vfsmount
  - 内核空间的路径path： vfsmount + dentry
  - 路径查找上下文：nameidata

13 文件打开过程分析(下)

- 源码分析

fs/open.c/do_sys_open->do_sys_openat2(dfd, filename, &how)
-> fd = get_unused_fd_flags(how->flags); //申请文件描述符
-> struct file *f = do_filp_open(dfd, tmp, &op); //创建file对象表示打开的文件
-> fd_install(fd, f); //将file和fd关联起来，file对象添加到fd_array数组中
最后返回fd给用户空间

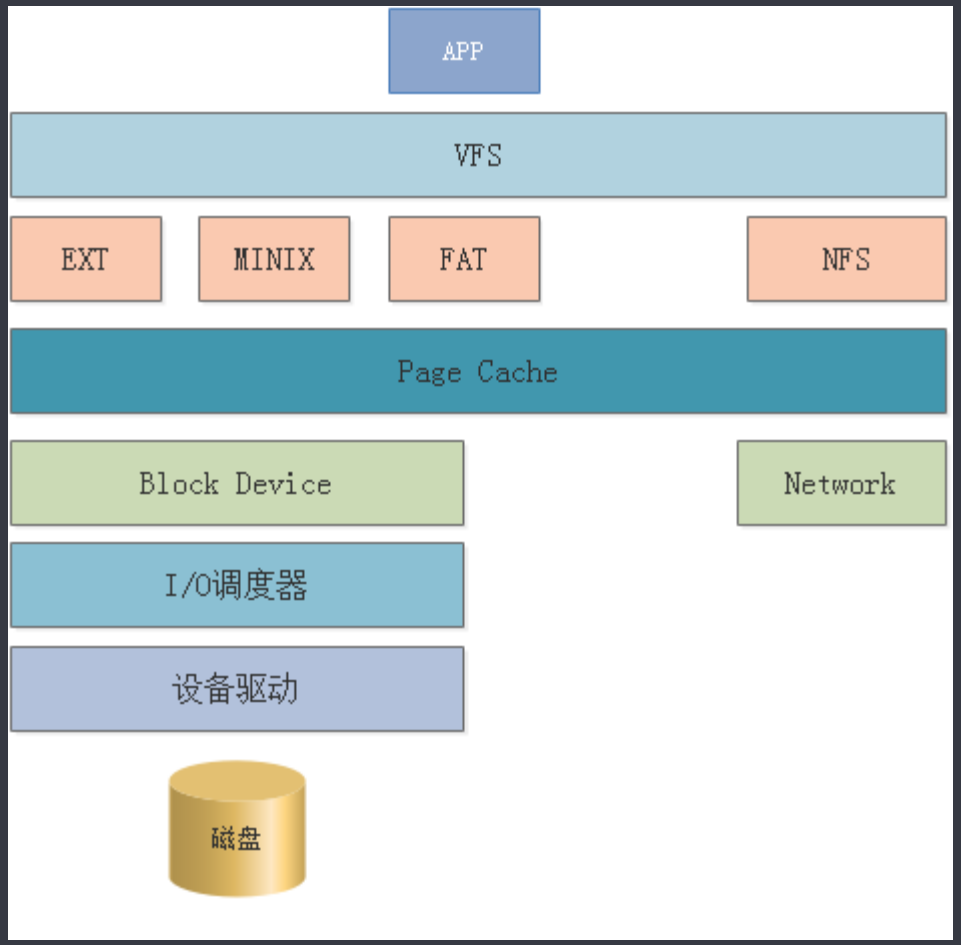
14 文件创建过程分析

- 文件的分类
  - 普通磁盘文件：open、close
  - 目录文件：mkdir、rmdir
  - 链接文件：symlink、unlink
  - 字符设备文件：mknod、unlink
  - 块设备文件：mknod 、unlink
  - 管道文件： mknod、unlink
  - 套接字文件： mknod、unlink

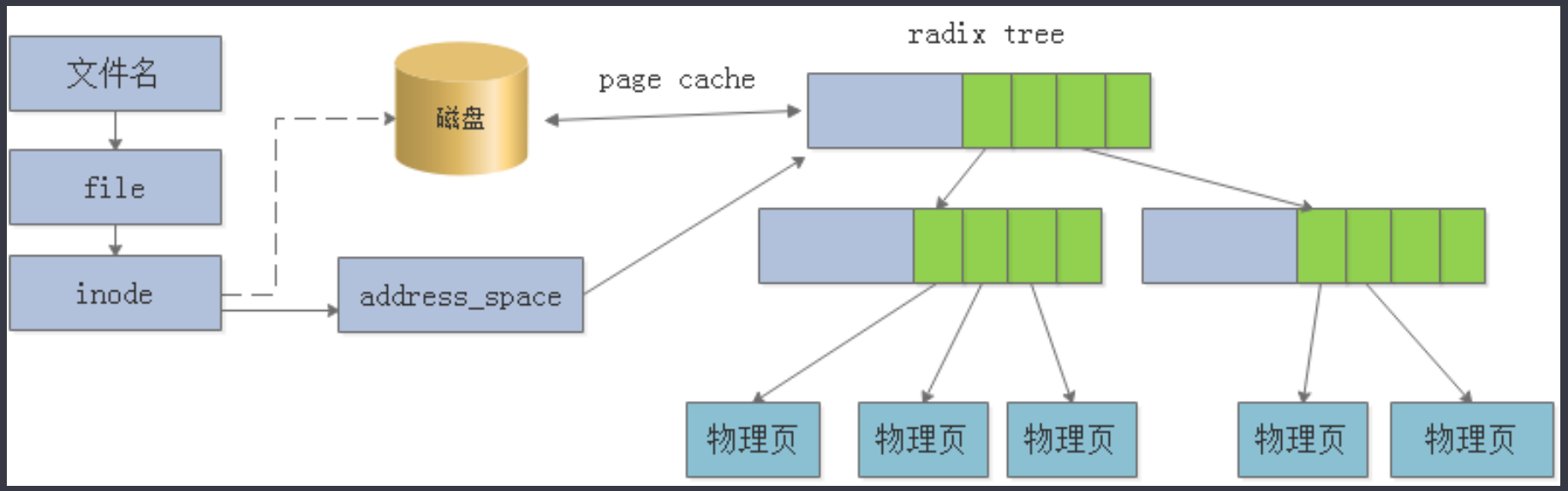
- 文件创建过程分析
  - 用户层：create\open\touch
  - 系统调用：do_sys_open
  - VFS层的操作：inode
  - 具体文件系统：minix
  - inode table、inode bitmap

15 文件读写过程（上）：地址空间与页缓存

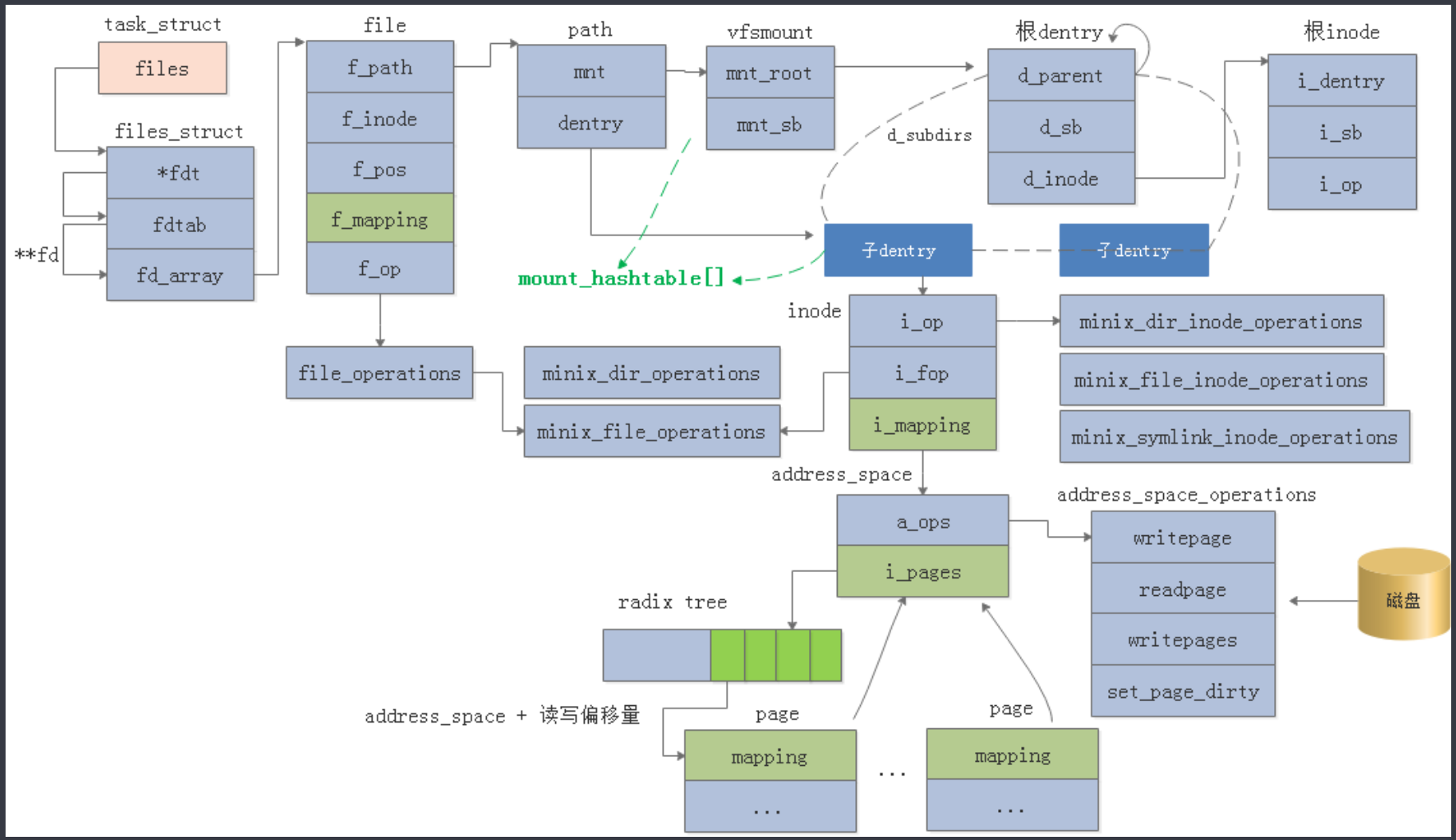
- page cache
- buffer cache



- 地址空间的概念
  - radix tree
  - address_space
  - address_space_operations



• 结构体关联



16 文件读写过程（下）：read内核流程分析

- read过程分析
  - 系统调用传参：文件描述符+文件位置偏移
  - 定位：fd-file-inode-address_space-page
  - 若找到对应的page，数据拷贝到用户空间
  - 若没找到page，内核新建一个page加入页缓存树中，并将数据从磁盘读到page cache
  - 将page上的数据拷贝到用户空间

  - 演示：minix 文件系统内核源码分析

17 什么是设备文件？

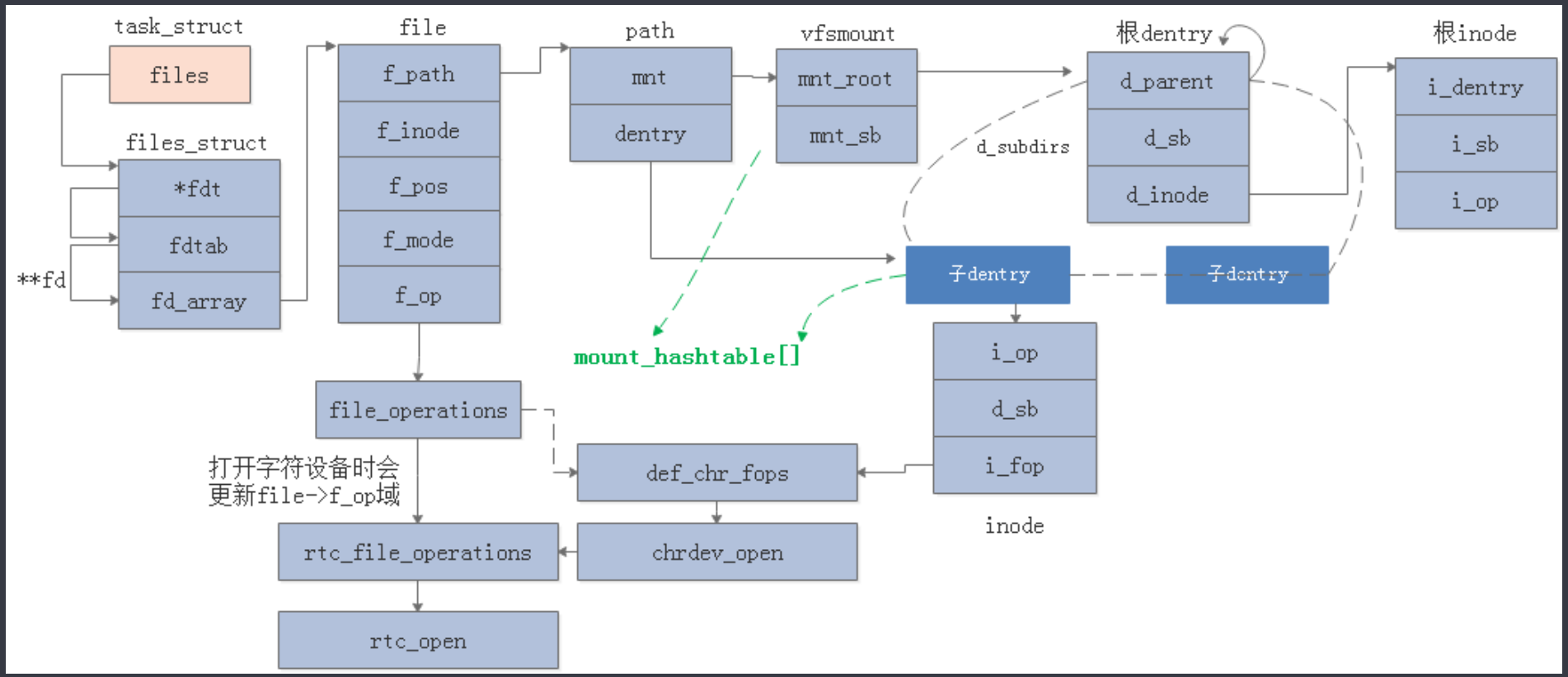
- 设备节点(文件)
  - 从文件系统的角度看设备文件
    - 字符设备是什么？
    - 块设备是什么？
    - 设备节点是什么？
    - 设备节点是怎么生成的？
    - 生成设备节点的三种方式:udev/mdev/mknod/内核创建
    - 文件系统：devtmpfs、tmpfs、/dev

18 设备文件创建过程分析

- 学习内容
  - 设备文件创建： mknod /dev/rtc1 c 255 1--inode
  - mknod流程分析
  - inode默认的file_operations
  - 创建过程中设备文件和普通文件的差异

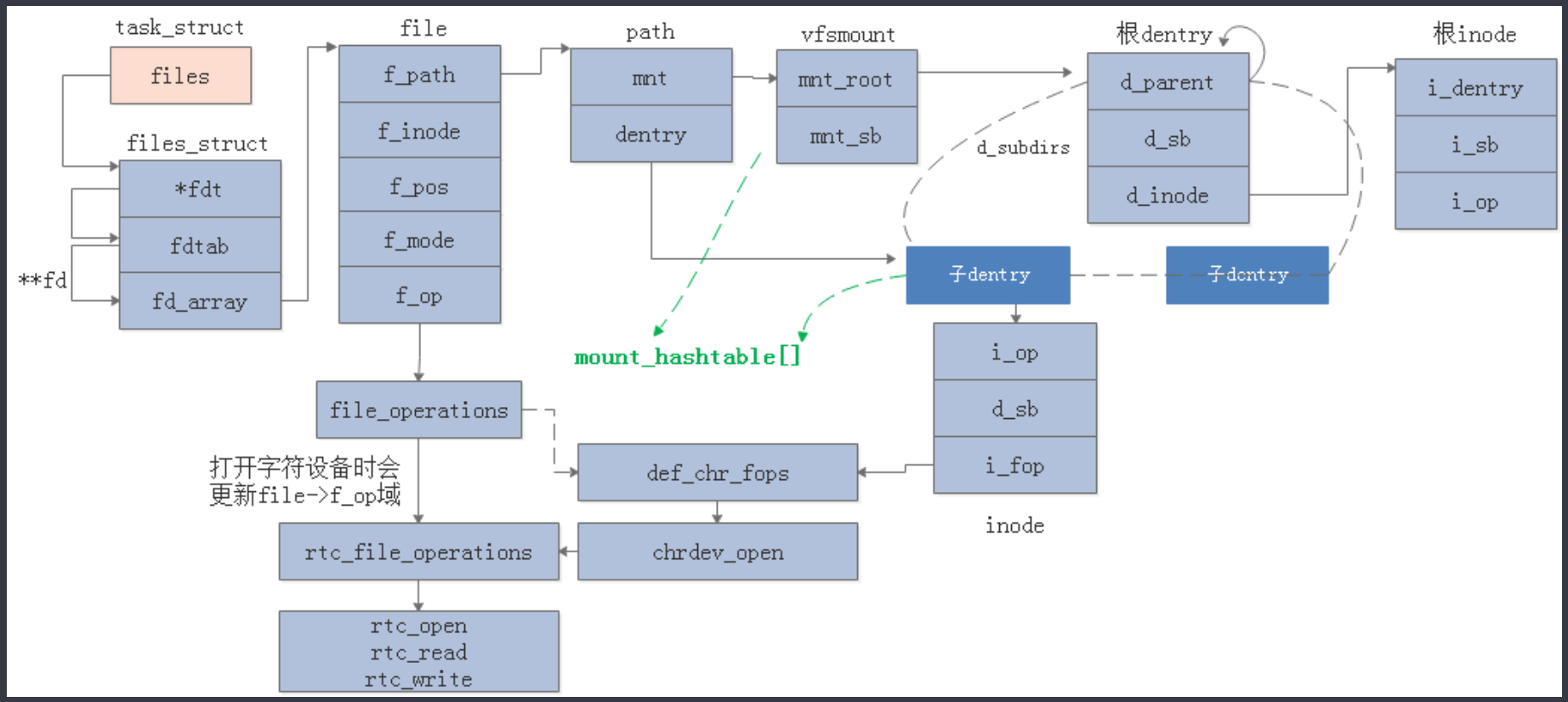
19 设备文件打开过程分析

- 设备文件的打开过程
  - 创建过程：
    - inode->i_fop = &def_chr_fops
    - inode->i_rdev = dev_t



20 设备文件读写过程分析

• 字符设备的读写过程


task_struct
files
files_struct
*fdt
fdtab
fd_array
**fd
file
f_path
f_inode
f_pos
f_mode
f_op
path
mnt
dentry
vfsmount
mnt_root
mnt_sb
根dentry
d_parent
d_sb
d_inode
d_subdirs
根inode
i_dentry
i_sb
i_op
子dentry
子dentry
mount_hashtable[]
i_op
d_sb
i_fop
inode
file_operations
打开字符设备时会
更新file->f_op域
def_chr_fops
chrdev_open
rtc_file_operations
rtc_open
rtc_read
rtc_write

21 什么是根文件系统？

- 根文件系统的基本概念
  - 什么是根文件系统？
  - 根文件系统和一般文件系统有什么区别？
  - 为什么需要根文件系统？
  - 根文件系统的目录
  - VFS和根文件系统是什么关系？
  - 如何指定根文件系统？
  - 挂载根文件系统的三种方式

• 挂载根文件系统的三种方式

```
How does it work?
=================

The kernel has currently 3 ways to mount the root filesystem:

a) all required device and filesystem drivers compiled into the kernel, no
   initrd.  init/main.c:init() will call prepare_namespace() to mount the
   final root filesystem, based on the root= option and optional init= to run
   some other init binary than listed at the end of init/main.c:init().

b) some device and filesystem drivers built as modules and stored in an
   initrd.  The initrd must contain a binary '/linuxrc' which is supposed to
   load these driver modules.  It is also possible to mount the final root
   filesystem via linuxrc and use the pivot_root syscall.  The initrd is
   mounted and executed via prepare_namespace().

c) using initramfs.  The call to prepare_namespace() must be skipped.
   This means that a binary must do all the work.  Said binary can be stored
   into initramfs either via modifying usr/gen_init_cpio.c or via the new
   initrd format, an cpio archive.  It must be called "/init".  This binary
   is responsible to do all the things prepare_namespace() would do.

   To maintain backwards compatibility, the /init binary will only run if it
   comes via an initramfs cpio archive.  If this is not the case,
   init/main.c:init() will run prepare_namespace() to mount the final root
   and exec one of the predefined init binaries.

Bryan O'Sullivan <bos@serpentine.com>
```

22 根文件系统挂载(上)：rootfs

- 学习内容
  - rootfs文件系统的概念
  - rootfs的作用
  - 为什么不直接挂载分区，而先使用rootfs？
  - rootfs和根文件系统是什么关系？
  - rootfs和根目录的关系？
  - 第一个根目录”/”何时创建的？

23 根文件系统挂载(中)：root=参数分析

- 学习内容
  - root=/dev/mmcblk0 挂载SD 卡为根文件系统
  - root=/dev/nfs 挂载NFS为根文件系统
  - noinitrd启动参数
  - init=/linuxrc启动

24 根文件系统挂载(下)：根目录 /

- 根目录：/
  - 作用是什么？
  - 什么时候初始化的？
  - 根文件系统和根目录是什么关系？
  - rootfs的根目录： /
  - NFS的根目录：/root-> vfsmount->mnt_root
  - 一个系统中允许多个”/”吗？
  - 根目录可以切换吗？

25 使用initrd作为根文件系统(上)

- 学习内容
  - 什么是 ramdisk？
  - 什么是initrd?
  - 如何制作initrd 镜像？
    - 通过ramdisk
    - 通过lookback
    - cpio格式：find . | cpio -o -H newc | gzip -c > ../initrd.gz
  - 如何设置root、initrd参数挂载initrd？
    - root=
    - initrd=
    - rdinit=/linuxrc

26 使用initrd作为根文件系统(中)：CPIO格式

- 学习内容
  - CPIO镜像：
  - find . | cpio --quiet -H newc -o | gzip -9 -n > ../initrd.gz
  - CPIO initrd 挂载过程分析
  - rdinit=/linuxrc的启动

27 使用initrd作为根文件系统(下)：ramdisk格式

- 学习内容
  - 制作一个Ramdisk镜像
  - Ramdisk格式的 initrd 挂载过程分析
  - 使用initrd的优点、缺点: page cache\地址空间

```
# dd if=/dev/zero of=initrd bs=1024 count=4096
# mkfs.ext2 –j initrd
# mount initrd /mnt
# cp –r /home/ramdisk/* /mnt
# umount /mnt
```

28 使用initramfs作为根文件系统

- 学习内容
  - 相比initrd，initramfs的优势
  - 启动参数
    - root：initramfs作为根文件系统，root、init数不用设置，只需要指定文件系统的源目录就可以了
    - 文件系统中默认要有一个init文件
  - 使用initramfs的内核配置
  - 使用initramfs的U-boot编译配置

```
[*] Enable a default value for bootcmd
tftp 0x60003000 uImage;tftp 0x60800000 vexpress-v2p-ca9.dtb;
setenv bootargs 'console=ttyAMA0';bootm 0x60003000 - 0x60800000;
```

29 initramfs挂载过程分析

- 学习内容
  - initramfs的挂载过程分析
  - Linkscript: arch/arm/kernel/vmlinux.lds
  - Head file: /include/asm-generic/vmlinux.lds.h
  - Makefile: /usr/Makefile

30 基于内存的文件系统：tmpfs

- 学习内容
  - 基于内存的文件系统
  - ramfs VS ramdisk
  - ramfs VS tmpfs
  - tmpfs的优点：
    - 快、不用格式化、动态调整大小、扩展性好
  - tmpfs的实现： mm/shmem.c
  - tmpfs的应用领域
    - 临时目录、临时文件
  - tmpfs的挂载
    - # mount -t tmpfs -o size=4m tmpfs /tmp

31 文件系统的自动挂载

- 学习内容
  - 文件系统挂载方式
    - 内核挂载、启动参数、自动挂载
  - 根文件系统的常用目录
  - 启动脚本
  - 文件系统的自动挂载

32 文件系统的性能指标

- 学习内容
  - 不同应用场景
    - 分布式
    - 服务器、PC
    - 消费电子
  - 文件系统的性能指标
    - 文件名长度、最大文件数、最大分区、最大文件大小
    - 挂载速度、读写速度、稳定性、日志
  - 如何选型文件系统？
  - 常见的文件系统
    - Windows：FAT16、FAT32、NTFS
    - Linux：ext2、ext3、ext4、btrfs、zfs
    - 光盘：ISO9660
    - 网络文件系统：SMBAFS、CIFS、NFS

更多信息

王利涛老师个人店：https://wanglitao.taobao.com
嵌入式在线教程网：www.zhaixue.cc
嵌入式技术交流群：
宅学部落02群：398294860
宅学部落03群：559671596
宅学部落04群：528718820

欢迎关注公众号：