東北大學秦皇島分校
Northeastern University at Qinhuangdao

Linux系统与内核分析

--文件系统

于七龙

目录

Part 1 Linux文件系统

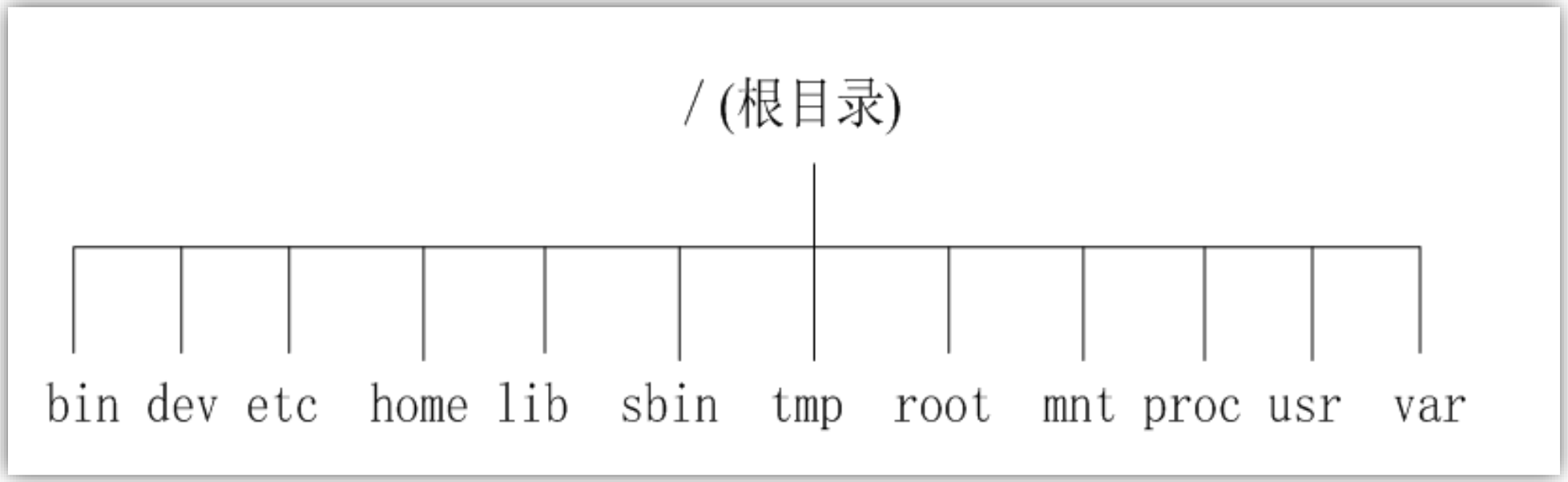
Part 2 虚拟文件系统

Part 3 非持久存储文件系统

Part 1 Linux文件系统

Linux文件结构

- Linux系统目录是树状结构
- 固定的目录规划有助于对系统文件和不同的用户文件进行统一管理



Linux目录树结构

Linux目录结构

- Linux是典型的树形文件结构

目录	说明	目录	说明
/	根目录，一切路径的起点	/etc	配置文件
/bin	可执行文件，如用户命令	/var	可变文件
/dev	设备文件	/home	普通用户home目录
/lib	库文件，内核模块文件	/root	root用户home目录
/proc	虚拟文件系统，内核映射文件	/opt	第三方软件
/sys	虚拟文件系统，设备相关映射文件	/sbin	重要的系统执行文件
/usr	继承于UNIX，存放程序与相关数据		
/boot	系统启动相关文件		

Linux文件类型

- Linux文件类型主要有
  - 常规文件
  - 目录文件
  - 设备文件
  - 管道文件
  - 链接文件

Linux文件类型

- 常规文件
  - 计算机用户和操作系统用于存放数据、程序等信息的文件，一般长期存放于外存设备中
  - 常规文件一般分为文本文件和二进制文件
- 目录文件
  - Linux文件系统将文件索引节点号和文件名同时保存于目录中，所以，目录文件就是将文件名称和其索引号结合在一起的一张表。目录文件只允许系统进行修改，用户进程可以读取目录文件，但不能对其进行修改

Linux文件类型

- 设备文件
  - Linux把所有的外设都当做文件来对待，每一种I/O设备对应一个设备文件，存放于/dev目录中
- 管道文件
  - 主要用于在进程间传递数据，管道是进程间传递数据的“媒介”。某进程数据写入管道的一端，另一个进程从管道另一端读取数据。
  - Linux对管道的操作与文件操作相同，管道文件又称为先进先出 (FIFO)文件

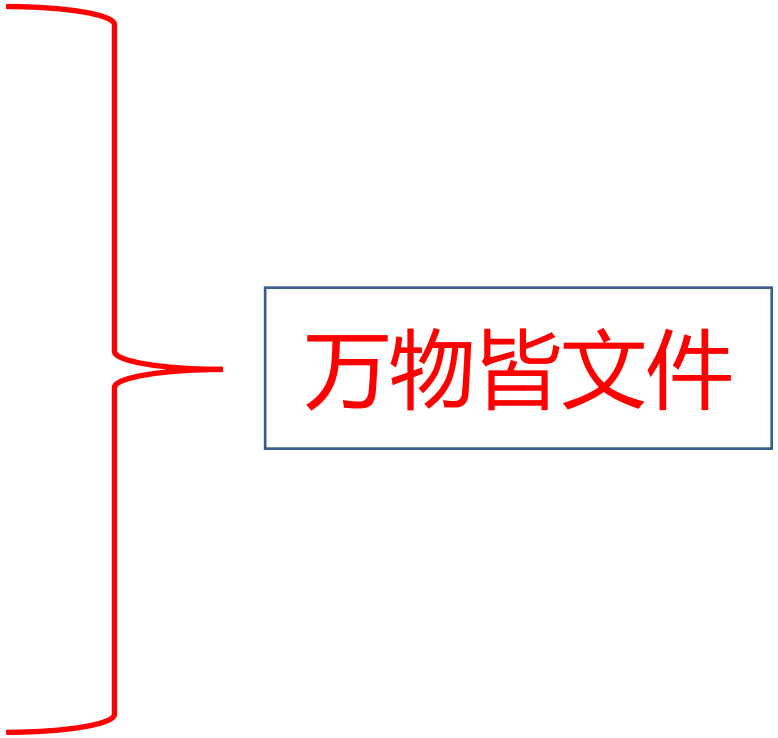
Linux文件类型

链接文件

- 链接文件提供了共享文件的一种方法，在链接文件中不是通过文件名实现文件共享，而是通过链接文件中包含的指向文件的指针实现对文件的访问
- 使用链接文件可以访问常规文件、目录文件等各类文件

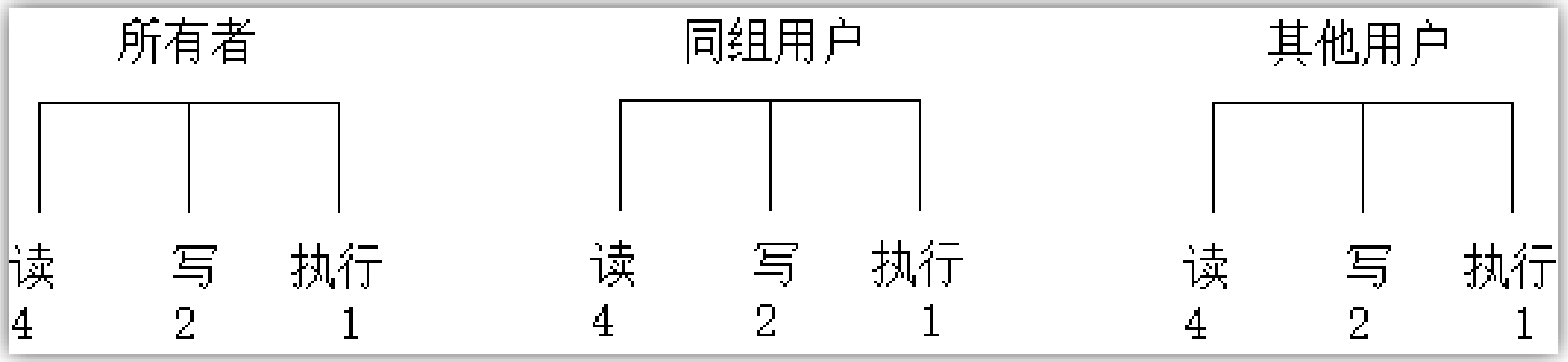
Linux文件类型

- Linux文件类型主要有
  - 常规文件
  - 目录文件
  - 设备文件
  - 管道文件
  - 链接文件



Linux文件权限

- Linux文件权限用于保证文件信息的安全，当文件被访问时，系统首先检验访问者的权限，只有与文件的访问权限相符时才允许对文件进行访问
- Linux针对文件所有者、与文件所有者同组的用户、其他用户设定了三级权限



Linux文件权限

Linux文件系统

- 文件系统指文件存在的物理空间，Linux系统中的每个分区都是一个文件系统，都有其自己的目录层次结构
- Linux会将分属不同分区、单独的文件系统按一定的方式形成一个系统的总的目录层次结构

索引节点

- Linux文件系统使用索引节点记录文件信息
  - 索引节点是一个数据结构，包含文件长度、创建时间、修改时间、权限、所属关系、磁盘中的位置等信息
- 每个文件或目录都对应一个索引节点，系统给每个索引节点分配一个号，称为**索引节点号**，文件系统通过索引节点号识别文件
  - 查看文件索引节点号：ls -i

链接

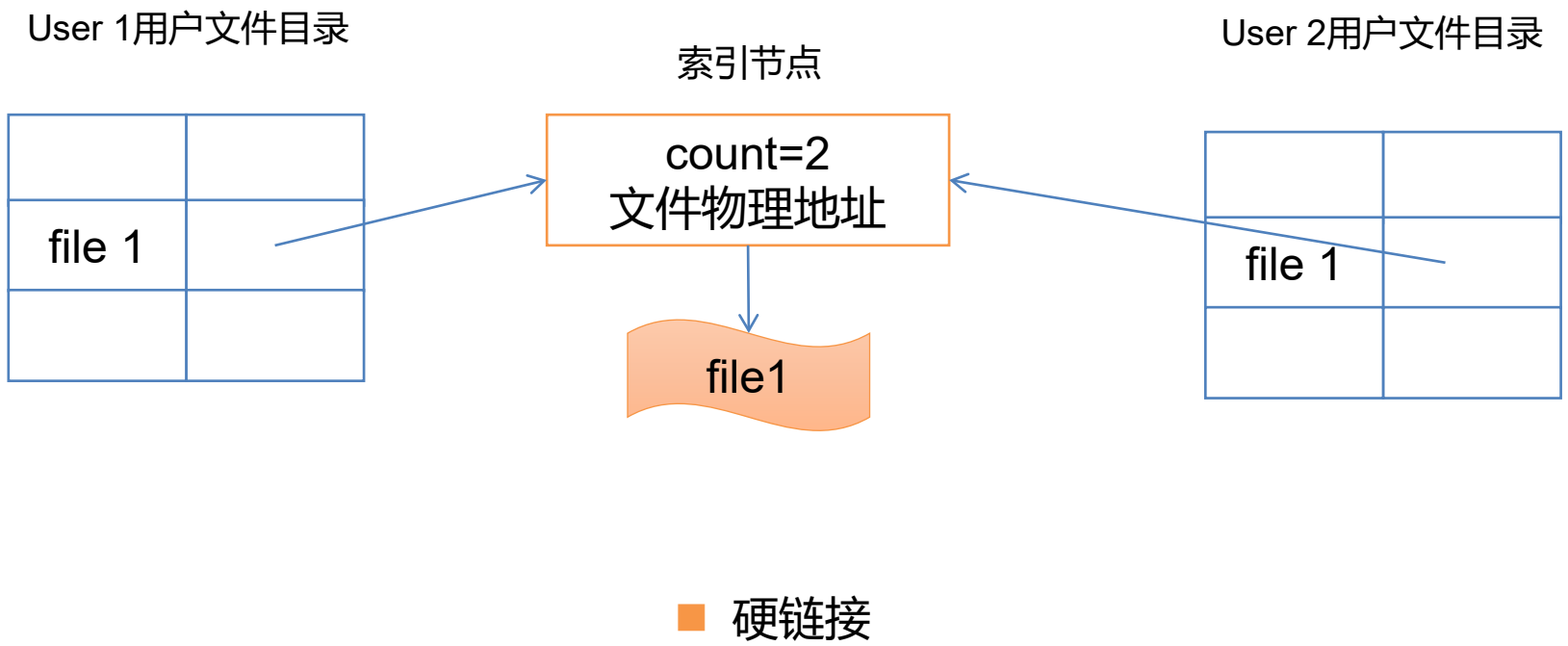
- Linux中可通过ln命令对一个已经存在的文件建立一个链接，而不复制文件的内容
- Linux中链接分为硬链接和软链接

```
root@bogon:/test# touch test1.c
root@bogon:/test# ln test1.c test2.c
root@bogon:/test# ln -s test1.c test3.c
root@bogon:/test# ls -i
24 test1.c  24 test2.c  26 test3.c
root@bogon:/test# ls -l
total 0
-rw-r--r-- 2 root root 0 May 31 22:39 test1.c
-rw-r--r-- 2 root root 0 May 31 22:39 test2.c
lrwxrwxrwx 1 root root 7 May 31 22:40 test3.c -> test1.c
```

硬链接与软链接

硬链接

- 硬链接是让一个文件对应多个文件名，文件名可在不同的目录中



硬链接

- 硬链接是让一个文件对应多个文件名，文件名可在不同的目录中
  - 硬链接中不同文件名其实是同一个文件，索引节点号相同
  - 一个文件有几个文件名，即称该文件的链接数为几
  - 硬链接有两个限制，一是不允许给目录创建硬链接，二是只有在同一文件系统中的文件之间才能创建链接
  - **若用户不再需要某硬链接文件，也不能删除该文件**

软链接

- 软链接也称符号链接
  - 软链接其实是链接文件，文件包含了另一个文件的路径名
  - 该路径名可指向位于任意一个文件系统的任意文件和目录，甚至可以指向一个不存在的文件
  - 删除符号链接不会破坏原有文件
  - 删除被链接文件会引起“被链接文件不存在”错误

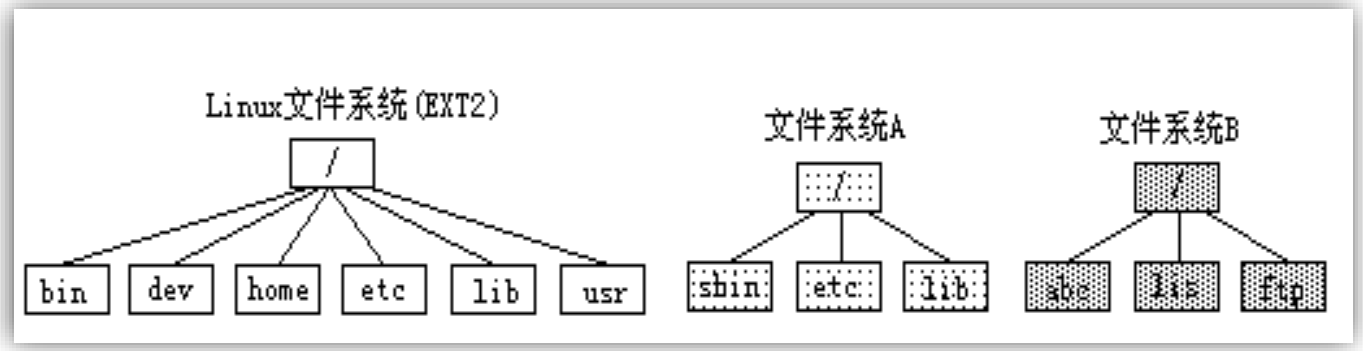
链接

- 链接操作为系统中已有的某个文件指定另外一个可访问该文件的名称，对于新的文件名，可为其指定不同的访问权限，以控制对信息的共享和安全性；如果链接指向目录，则可避免使用过长的路径名

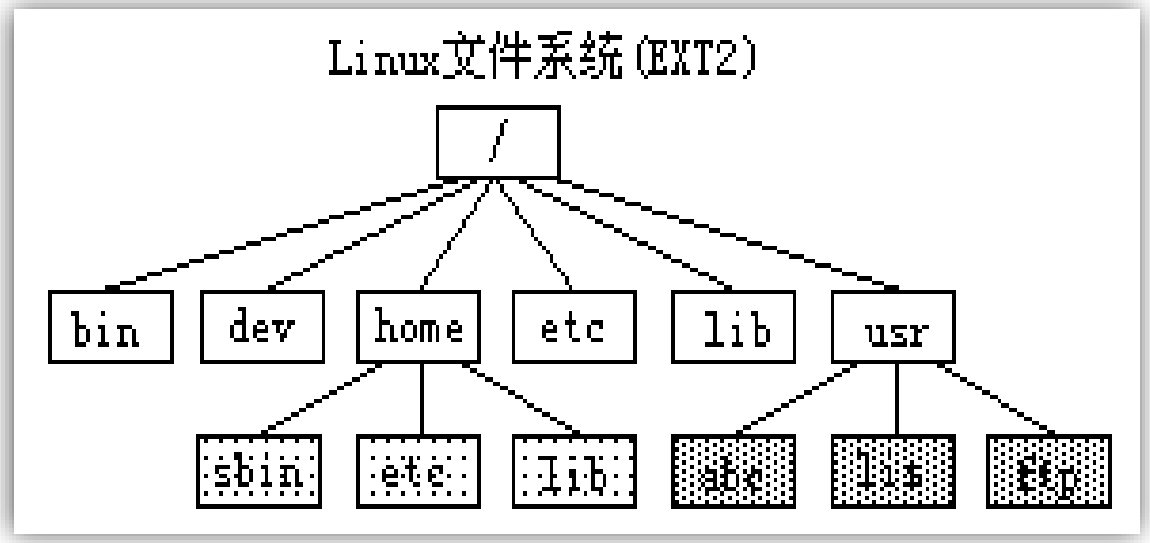
安装文件系统

- 将一个文件系统的顶层目录挂到另一个文件系统的子目录上，使之成为一个整体，称为安装，该子目录称为“安装点”
- 文件系统的安装俗称为“挂载（mount）”
  - 安装文件系统使用mount命令

安装文件系统



- 安装前三个独立的文件系统



- 安装后的文件系统

Linux支持的文件系统

- 内核源代码fs/目录，包含Linux所支持的所有文件系统的实现
- 可通过df命令查看系统已挂在的分区及对应的文件系统

```
root@bogon:/# df -T
Filesystem     Type     1K-blocks     Used Available Use% Mounted on
udev           devtmpfs    937476        0    937476   0% /dev
tmpfs          tmpfs       194496     8480    186016   5% /run
/dev/sda1      ext4      18447056 12451536   5035420  72% /
tmpfs          tmpfs       972468     1044    971424   1% /dev/shm
tmpfs          tmpfs         5120        0      5120   0% /run/lock
tmpfs          tmpfs       972468        0    972468   0% /sys/fs/cgroup
tmpfs          tmpfs       194492       12    194480   1% /run/user/0
tmpfs          tmpfs       194492        8    194484   1% /run/user/129
tmpfs          tmpfs       194492        0    194492   0% /run/user/1000
```

系统已挂载的分区及文件系统

Part 2 虚拟文件系统

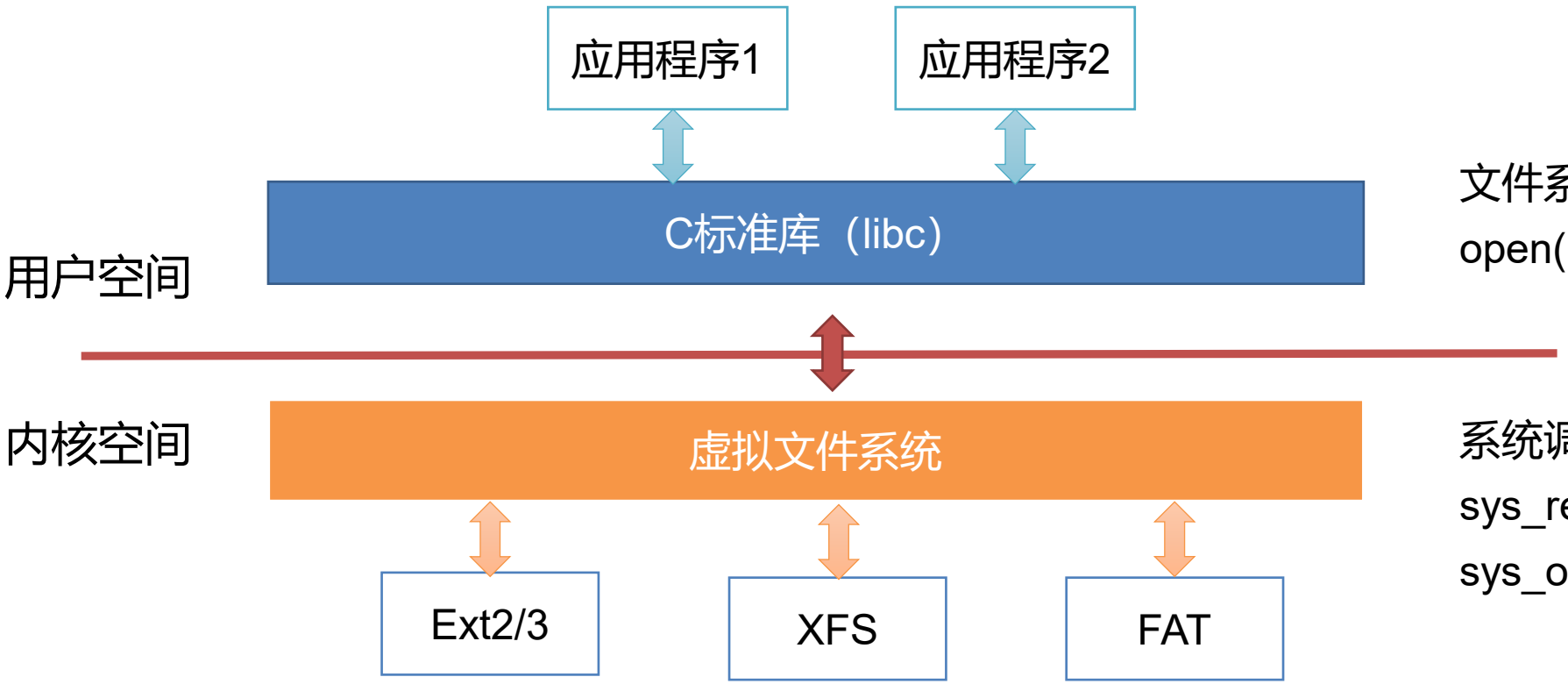
虚拟文件系统

- 为了支持各种不同的文件系统，Linux提供了一种统一的框架，使得用户程序可以通过同一个文件系统界面，即同一组系统调用，能够对各种不同的文件系统及文件进行操作，这样，用户程序可以不关心各种不同文件系统的实现细节，而使用系统提供的统一、抽象、虚拟的文件系统界面。这种统一的框架称为虚拟文件系统转换（Virtual Filesystem Switch），简称虚拟文件系统(VFS)

虚拟文件系统

- 虚拟文件系统所提供的抽象界面主要由一组标准的、抽象的操作构成，如read()、write()、lseek()等，这些函数以系统调用的形式供用户程序调用，这样，用户程序调用这些系统调用时，无须关心所操作的文件属于哪个文件系统，以及该文件系统是如何设计和实现的

VFS与具体文件系统的关系



文件系统的API，如read()、write()、open()、close()等

系统调用对应的内核函数，如sys_read()、sys_write()、sys_open()、sys_close()等

■ VFS与具体文件系统的关系

VFS与具体文件系统的关系

- Linux的目录建立了一棵根目录为“/”的树，根目录包含在根文件系统中，Linux中根文件系统通常是ext*类型，其它所有文件系统都可以被“挂载”在根文件系统的子目录中

VFS的对象

- Linux文件系统的设计继承于UNIX
- VFS必须承载各种文件系统的共有属性，且只管理挂载到系统中的实际文件系统
- VFS包含4个对象
  - 超级块对象（superblock）
  - 索引节点对象（inode）
  - 目录项对象（dentry）
  - 文件对象（file）

VFS的对象

- Linux中文件是一个有序字节串
- 文件系统通过目录组织文件，目录通过层层嵌套形成文件路径，路径中的每部分称为目录项
  - 如/home/yql/lab/test.c，其中根目录是/，yql、lab是目录项，test.c是文件
  - UNIX/Linux中，目录也是文件的一种，所以对目录和文件可使用相同的操作
- 索引节点用于描述文件的属性（文件名、权限、大小、拥有者、创建时间等）
- 超级块是一种包含文件系统信息的数据结构

VFS的对象

- VFS对象总结
  - 超级块对象：描述已经安装的文件系统
  - 索引节点对象：描述一个文件
  - 目录项对象：描述一个目录项，是路径的组成部分
  - 文件对象：描述由进程打开的文件

因为VFS将目录作为一个文件来处理，所以不存在目录对象，或者说，**目录项不同于目录，目录却和文件相同**

超级块

- 超级块用于描述整个文件系统的信息，对于每个具体的文件系统，都有与之对应的超级块
- 文件系统存放于磁盘中，超级块只存在于内存中
  - 内核在对一个文件系统进行初始化和注册时在内存为其分配一个超级块，即超级块是各种具体的文件系统在安装时建立，并在文件系统卸载时被自动删除

超级块

超级块的数据结构

```
#include/linux/fs.h
struct super_block {
        struct list_head        s_list;                 //超级块链表
        dev_t                   s_dev;                  //具体文件系统的块设备标识符
        unsigned long           s_blocksize;            // 块大小
        loff_t                  s_maxbytes;             //文件最大长度/
        struct file_system_type   *s_type;              //具体的文件系统类型
        const struct super_operations *s_op;                    //超级块操作函数合集
        struct dentry           *s_root;                // 文件系统根目录地址
        struct rw_semaphore       s_umount;             // 卸载所用的信号量
        int                     s_count;
        atomic_t                s_active;
        struct list_head        s_inodes;               // 所有索引节点的链表
        ......
};
```

超级块

- 超级块操作表是与超级块关联的方法，每一种文件系统都有自己的

  super_operations操作实例

```
#include/linux/fs.h
struct super_operations {
        struct inode *(*alloc_inode)(struct super_block *sb);
        void (*destroy_inode)(struct inode *);
        void (*dirty_inode) (struct inode *, int flags);
        int (*write_inode) (struct inode *, struct writeback_control *wbc);
        int (*drop_inode) (struct inode *);
        void (*evict_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        ......
};
```

索引节点

- 索引节点代表存储设备上的实际物理文件，相当于文件的“档案”
- 文件名可以随时更改，但索引节点对文件是唯一的，并随着文件的存在而存在
- 索引节点分为静态索引节点和动态索引节点
  - 具体文件系统的索引节点是存放于磁盘上的静态结构，需要使用索引节点时，必须调入内存，填写VFS的索引节点，称为动态索引节点

索引节点

索引节点的数据结构

```
struct inode {
        unsigned long           i_ino         /* 索引节点号*/
        umode_t                 i_mode;         /* 文件访问权限*/
        kuid_t                  i_uid;          /* 文件拥有者标识号*/
        kgid_t                  i_gid;          /* 文件拥有者所在组标志号*/
        const struct inode_operations     *i_op;    /* 索引节点的操作函数集 */
        struct super_block      *i_sb;                  /* 所属文件系统超级块*/
        union {
                struct hlist_head i_dentry;          /* 所有引用该inode的目录项链表*/
                struct rcu_head         i_rcu;
        };
        struct address_space    *i_mapping;
        struct timespec         i_atime;          /* 最近访问时间*/
        struct timespec         i_mtime;         /* 最近修改时间*/
        struct timespec         i_ctime;         /* 最近一次修改inode时间*/
        spinlock_t        i_lock;            /* i_blocks, i_bytes, maybe i_size */
        ......
};
```

索引节点

- 在同一个文件系统中，每个索引节点号都是唯一的，内核根据索引节点号的哈希值查找其inode结构
  - 索引节点号是“局部变量”
  - 每个文件系统对应一个索引节点哈希表

索引节点

- 索引节点操作表是与索引节点关联的方法，不同文件系统中每个函数的具体实现不同

```
struct inode_operations {
        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);
        ......
}
```

目录项

- 每个文件除了有一个索引节点inode数据结构外，还有一个目录项dentry数据结构
- 一个有效的目录结构，必定有一个inode结构与之对应
- 一个inode可能对应多个目录结构

目录项

- 目录项只存在于内存
  - dentry结构代表逻辑意义上的文件，所描述的是文件逻辑上的属性，因此，目录项对象在磁盘上并没有对应的映像，而inode结构代表的是物理意义上的文件，记录的是物理上的属性
  - 基于以上原因，inode结构与dentry结构虽然都是对文件各方面属性进行描述，但在Linux系统中并没有合二为一

文件对象

- 文件对象用于描述被进程打开的文件
- file结构中主要保存了打开文件的文件位置
- 同一个物理文件可能存在多个对应的文件对象
- 文件对象只存在于内存

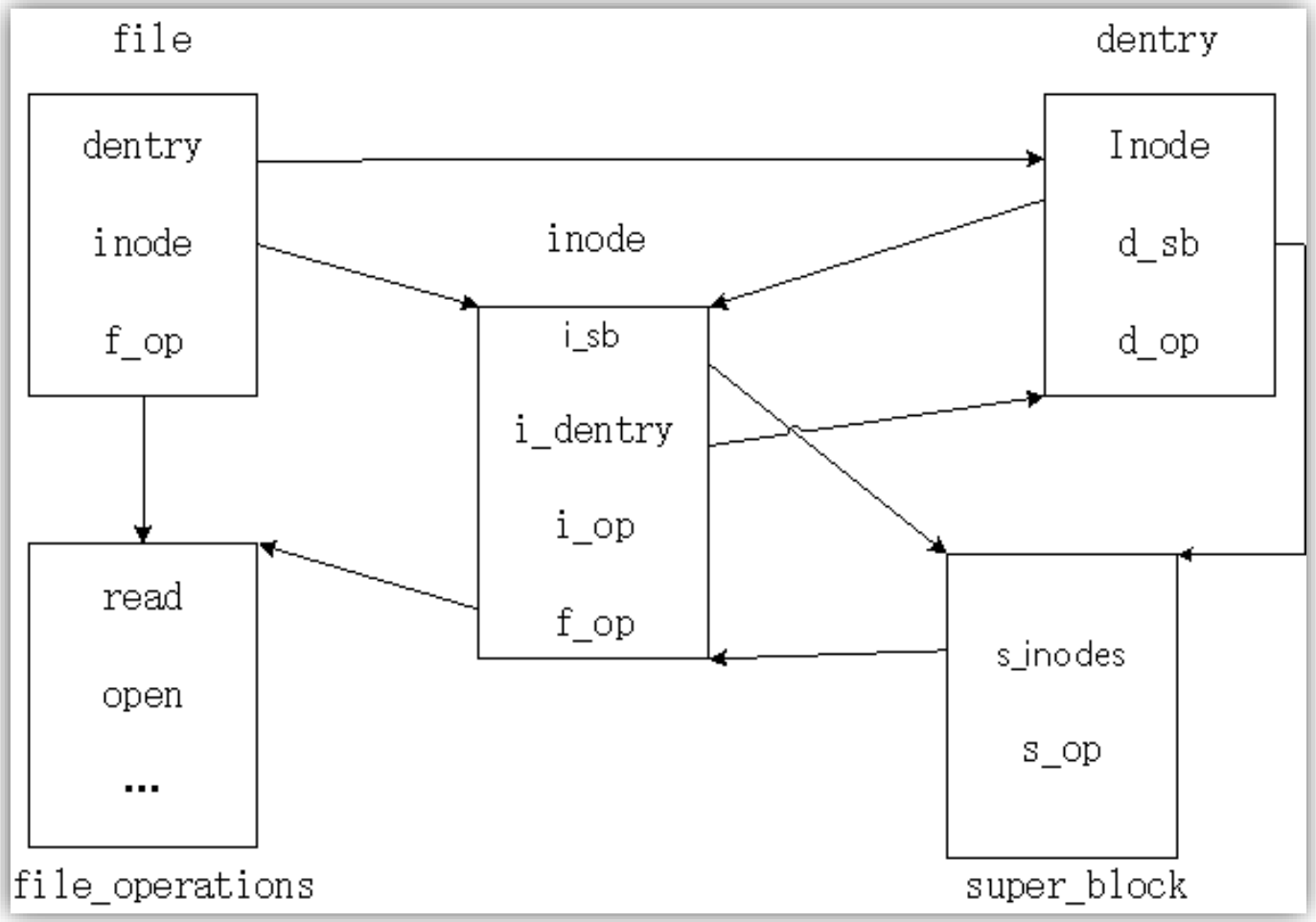
文件对象

- 文件操作表是对文件进行操作的一组函数

```
struct file_operations {
        struct module *owner;                                   //拥有声明
        loff_t (*llseek) (struct file *, loff_t, int);          //修改文件指针
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *); //从文件中按字节读取
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);      //向文件中按字节写入

        int (*mmap) (struct file *, struct vm_area_struct *);           //文件到内存映射
        int (*open) (struct inode *, struct file *);                    /* 打开文件
        int (*flush) (struct file *, fl_owner_t id);                    // 关闭文件时减少f_count计数
        int (*release) (struct inode *, struct file *);                 //释放file对象
        ……
};
```

VFS对象间的关系



- 主要数据结构间的关系

Part 3 非持久存储文件系统

非持久存储文件系统

- 非持久存储文件系统也称伪文件系统，只是为了使用虚拟文件系统的编程接口。常用非持久存储文件系统有
  - **proc文件系统**：Linux最初开发proc文件系统目的是把内核中的进程信息导出到用户空间，后来扩展到把内核中的任何信息导出到用户空间，常挂载于/proc下
  - **sysfs**:用于把内核的设备信息导出到用户空间，常把挂载于/sys下

proc文件系统

- 该类文件系统在内核中生成，且只存在于内存
- proc文件系统使内核可以生成与系统状态、配置等有关的信息
- 只有用户发出读操作请求时，才会生成相关信息