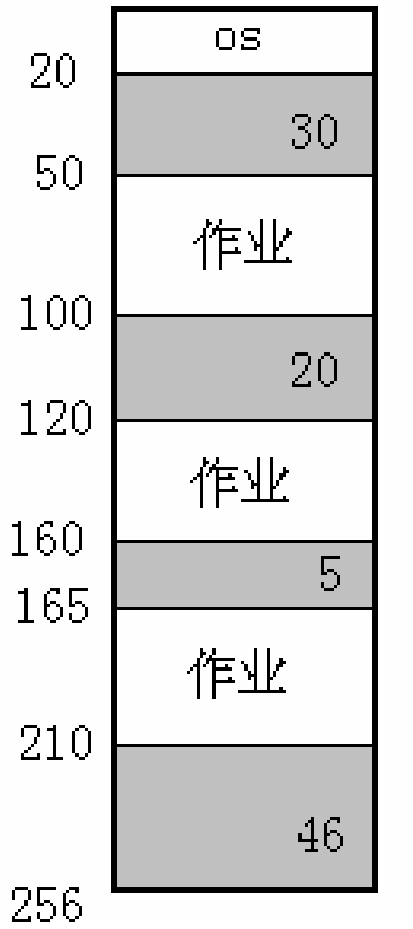
操作系统

L19 文件系统实现（上）

胡燕
大连理工大学 软件学院

连续内存分配

习题



- 请求表中的作业序列：
- （1） 作业A要求18K；作业B要求25K，作业C要求30K
- 请使用3种动态分区分配方法，为上述进程请求序列进行内存分配

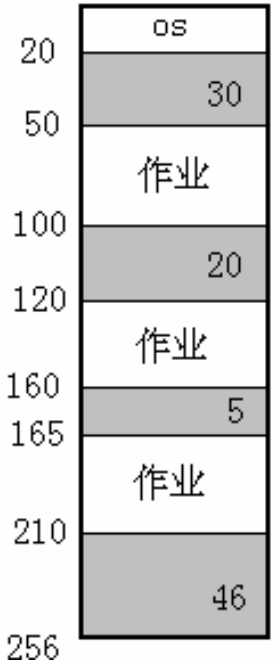
主观题 10分

设置

进程对内存的请求序列：

（1） 作业A要求18K；作业B要求25K，作业C要求30K

请使用3种动态分区分配方法（首次适配、最佳适配、最差适配），为上述进程序列进行内存分配。

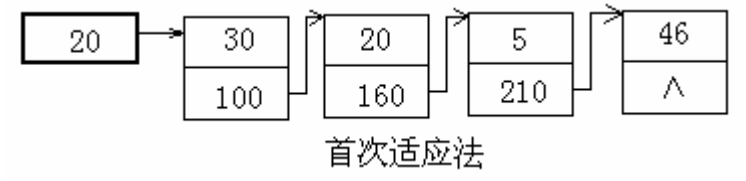


作答

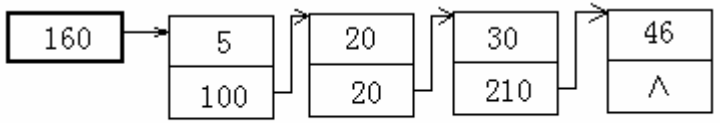
连续内存分配

习题

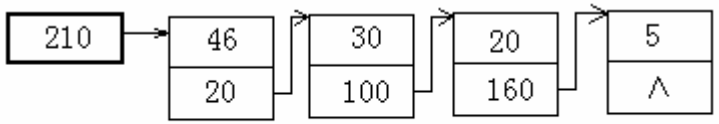
问题中的空闲链表组织（首次适应、最佳适应、最差适应）



首次适应法



最佳适应法



最坏适应法

空闲链表的最佳组织形式

Request: A(18K), B(25K),C(30K)

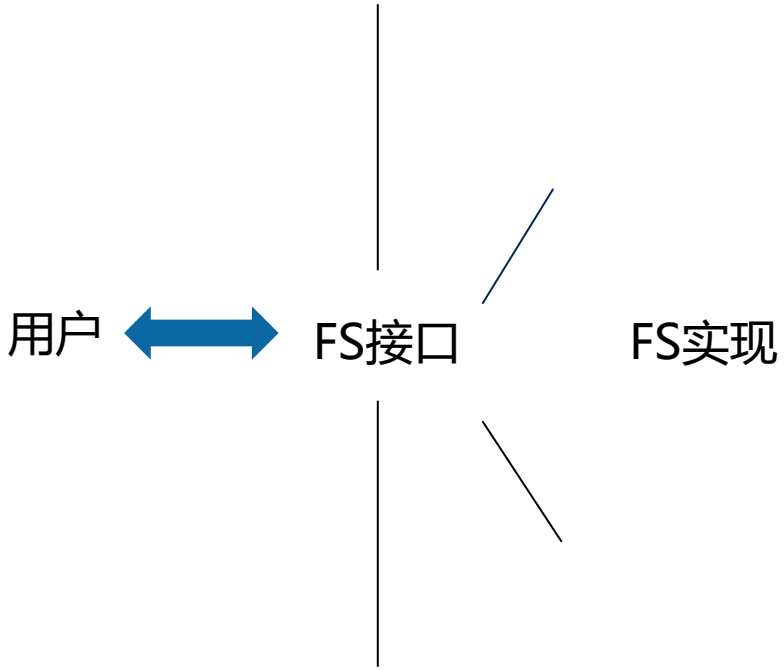
上讲回顾

文件系统接口

- 文件概念
- 文件操作
- 分区与目录结构
- 典型目录结构
- 文件系统加载与保护

接口->实现

文件系统实现



文件系统设计与实现

完成数据在外存的基本存储功能

早期的FS

- File Organization
- Directory Structure
- Access Control
- Performance
- File Backup and Recovery
- Scalability

现代FS所必须做的多方面考虑

文件系统设计

现代FS已经变得非常复杂的一个子系统，而且其设计好坏对OS性能有重要影响

文件系统设计者们对文件系统的复杂功能进行了梳理，通常采取分层的设计方式

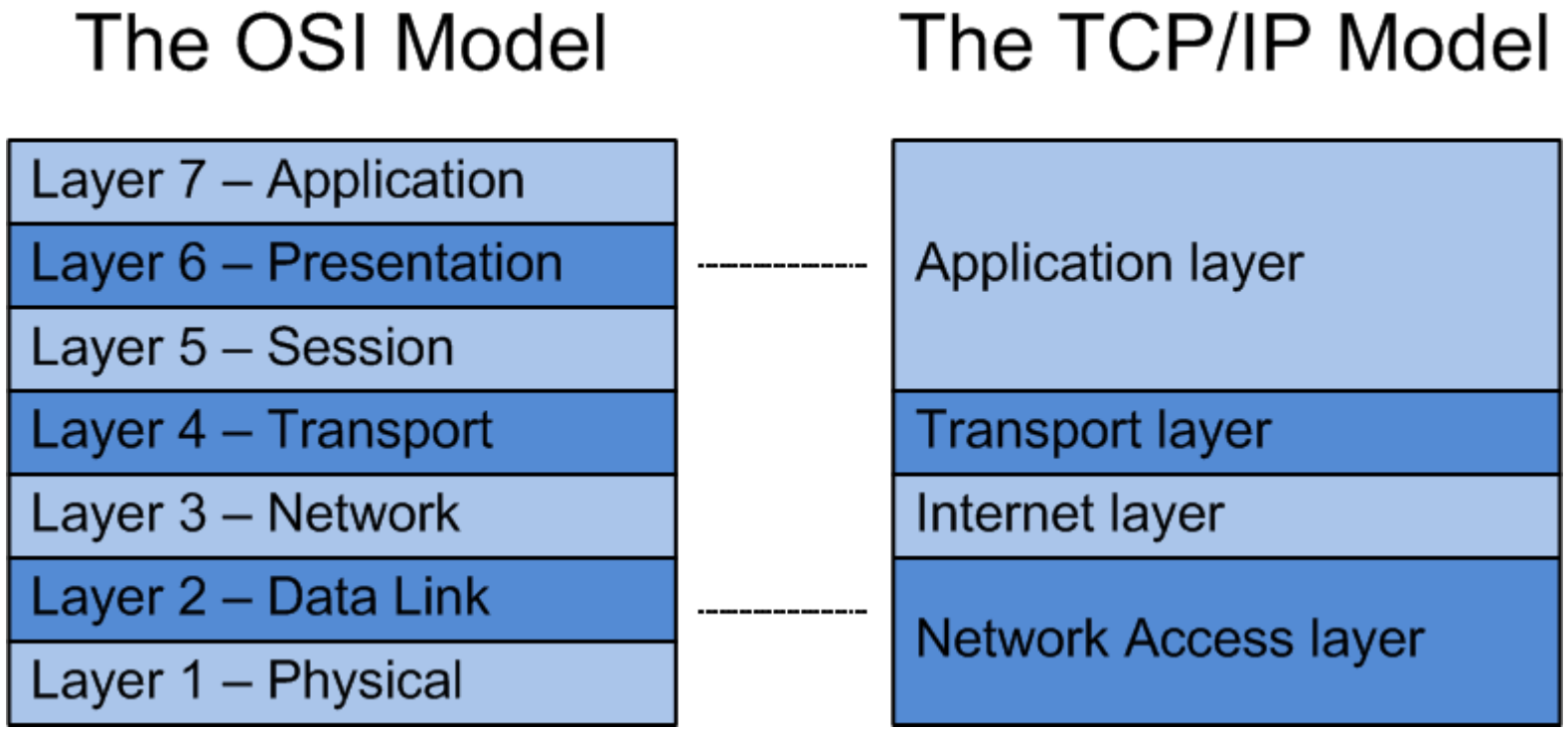
分层设计是面对软件复杂性时常用的一种设计模式

- Separation of concerns & modularity in the code
- Allows flexibility and scalability in the system

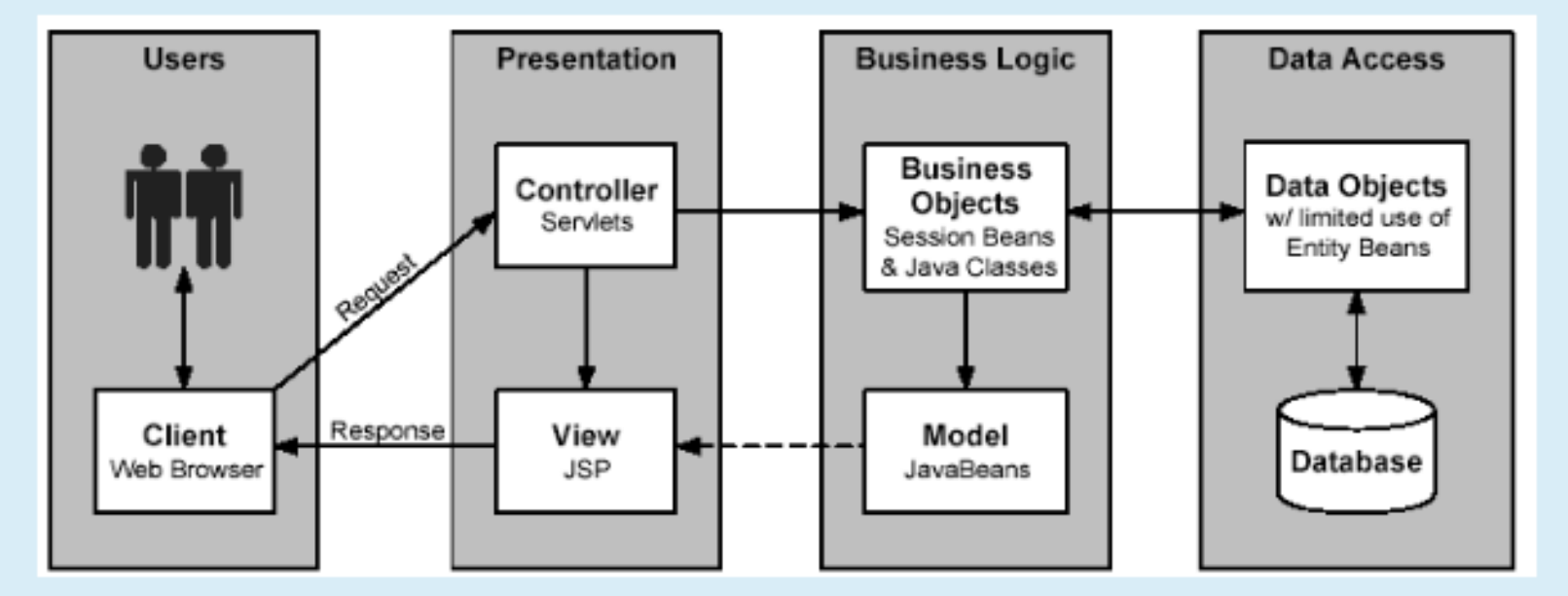
文件系统设计

各位同学目前印象最为深刻的分层软件设计案例是?

分层设计案例


The OSI Model
The TCP/IP Model
Layer 7 – Application
Layer 6 – Presentation
Layer 5 – Session
Layer 4 – Transport
Layer 3 – Network
Layer 2 – Data Link
Layer 1 – Physical
Application layer
Transport layer
Internet layer
Network Access layer

分层设计案例


Users
Presentation
Business Logic
Data Access
Controller
Servlets
Business Objects
Session Beans & Java Classes
Data Objects
w/ limited use of Entity Beans
Request
Client
Web Browser
Response
View
JSP
Model
JavaBeans
Database

文件系统设计

Coming next =>>>>>> 文件系统的分层设计

文件系统结构设计

File System Organization

01

文件系统设计需求

- 文件在哪个目录?
- 文件的元信息在何处?

- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?

- 如何与设备驱动层交互，进行实质的文件系统数据读写?

- 如何实现对读写过程的I/O控制?

文件系统设计需求

- 文件在哪个目录?
- 文件的元信息在何处?

- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?

- 如何与设备驱动层交互，进行实质的文件系统数据读写?

- 如何实现对读写过程的I/O控制?

文件系统分层设计示意图

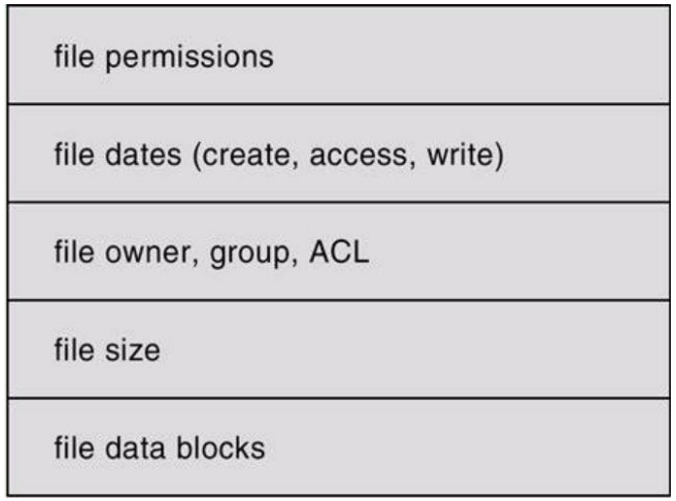
Application Programs

Logical File System

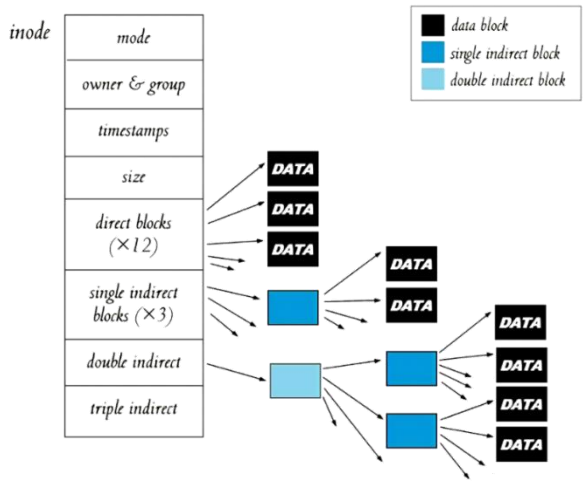
Q: 应用层面操作文件时，需要何种支持?

1.文件在哪个目录? → 管理目录结构

2.文件的元信息在何处? → 创建FCB、建立文件名到FCB的映射



FCB（File Control Block, 文件控制块）

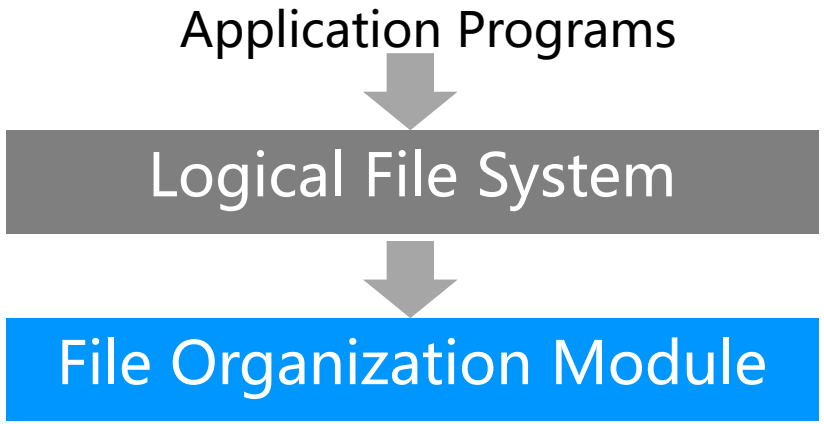


Unix inode

文件系统设计需求

- 文件在哪个目录?
- 文件的元信息在何处?
- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?
- 如何与设备驱动层交互，进行实质的文件系统数据读写?
- 如何实现对读写过程的I/O控制?

文件系统分层设计示意图



Q: 对文件组织重点需要考虑哪些方面?

1.如何为文件分配空间
2.如何对文件已分配空间进行组织管理

-建立文件的逻辑块与磁盘物理块之间的映射关系
-如何管理磁盘上的空闲物理块

文件系统设计需求

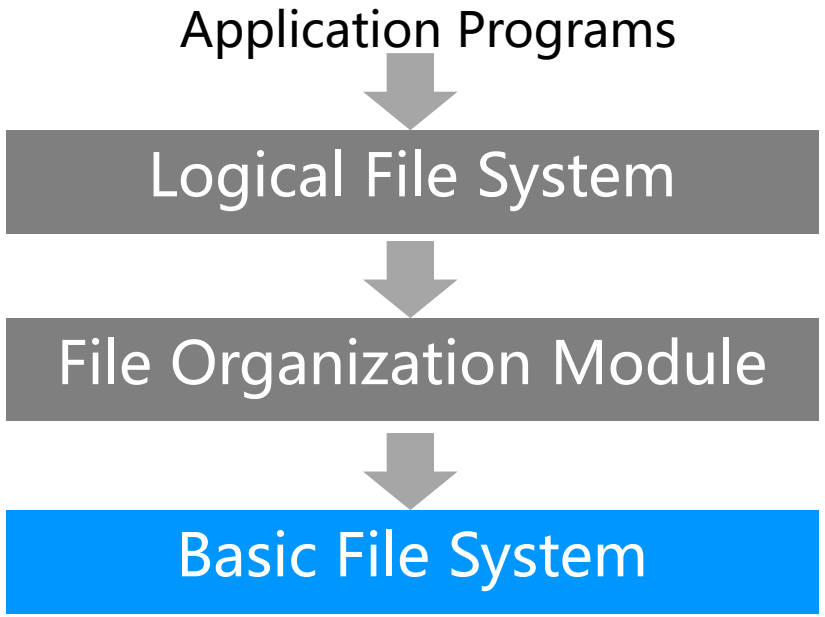
- 文件在哪个目录?
- 文件的元信息在何处?

- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?

- 如何与设备驱动层交互，进行实质的文件系统数据读写?

- 如何实现对读写过程的I/O控制?

文件系统分层设计示意图



与磁盘设备驱动程序对接，实施以块为单位的数据读写

文件系统设计需求

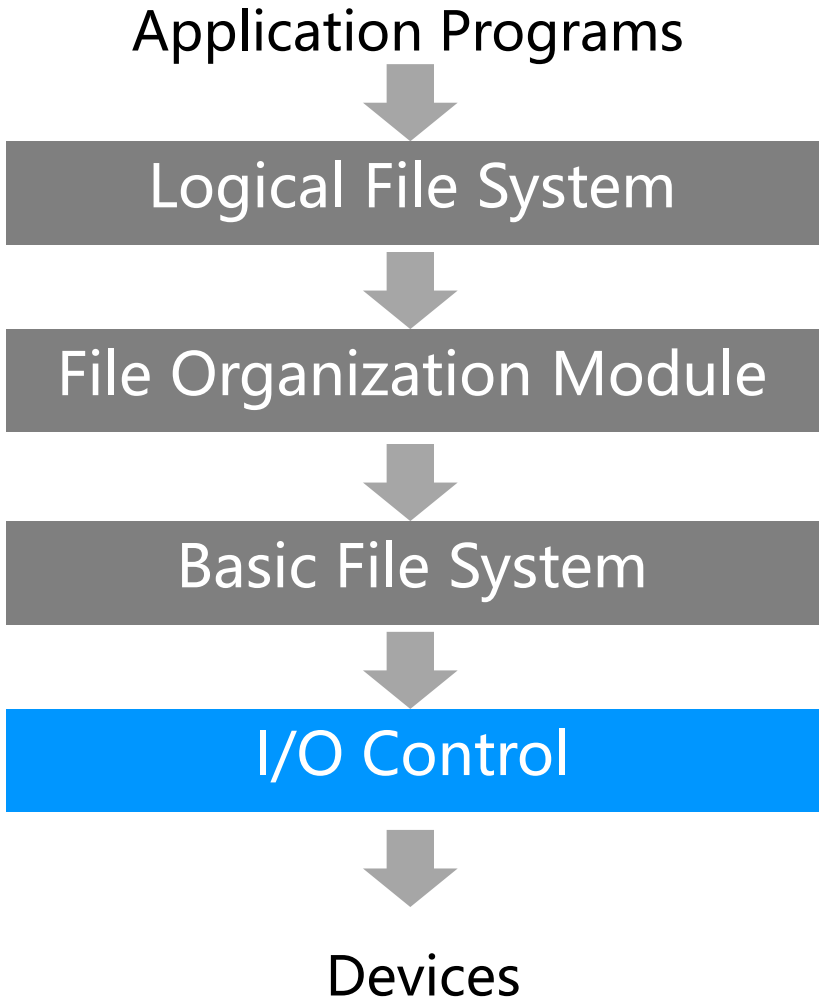
- 文件在哪个目录?
- 文件的元信息在何处?

- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?

- 如何与设备驱动层交互，进行实质的文件系统数据读写?

- 如何实现对读写过程的I/O控制?

文件系统分层设计示意图



I/O控制软件层:
磁盘设备控制器驱动是其核心

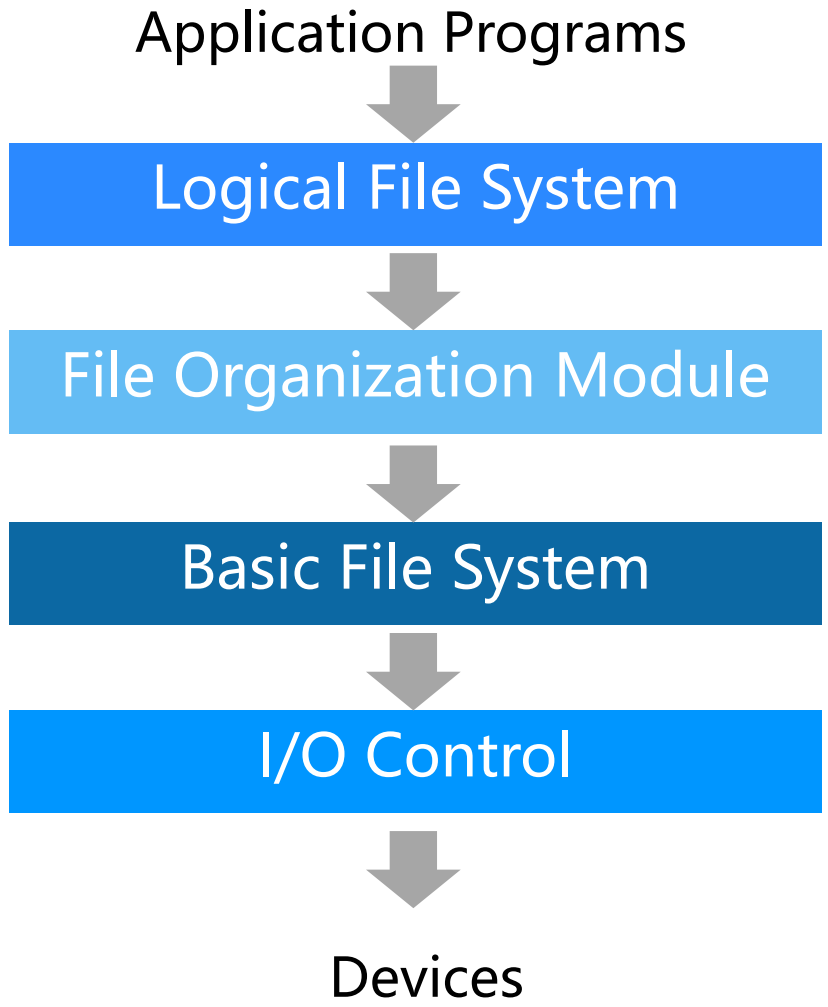
文件系统设计需求

- 文件在哪个目录?
- 文件的元信息在何处?

- 如何为文件分配空间?
- 如何对文件已分配空间进行组织管理?

- 如何与设备驱动层交互，进行实质的文件系统数据读写?

- 如何实现对读写过程的I/O控制?



What is Virtual File System

Virtual filesystems are the magic abstraction that makes the "everything is a file" philosophy of Linux possible.

What is Virtual File System

Virtual filesystems are the magic abstraction that makes the "everything is a file" philosophy of Linux possible.

```
root@a24671bde3dc:~# echo "hello world" > hello.txt
root@a24671bde3dc:~#
```

echo命令重定向到 普通文件 hello.txt，命令执行后在终端不输出任何信息

What is Virtual File System

Virtual filesystems are the magic abstraction that makes the "everything is a file" philosophy of Linux possible.

```
root@a24671bde3dc:~# echo "hello world" > /dev/console
hello world
root@a24671bde3dc:~#
```

Interesting! Why?

What is Virtual File System

Virtual filesystems are the magic abstraction that makes the "everything is a file" philosophy of Linux possible.

```
root@a24671bde3dc:~# echo "hello world" > /dev/console
hello world
root@a24671bde3dc:~#
```

echo命令重定向到 普通文件 hello.txt，命令执行后在终端不输出任何信息

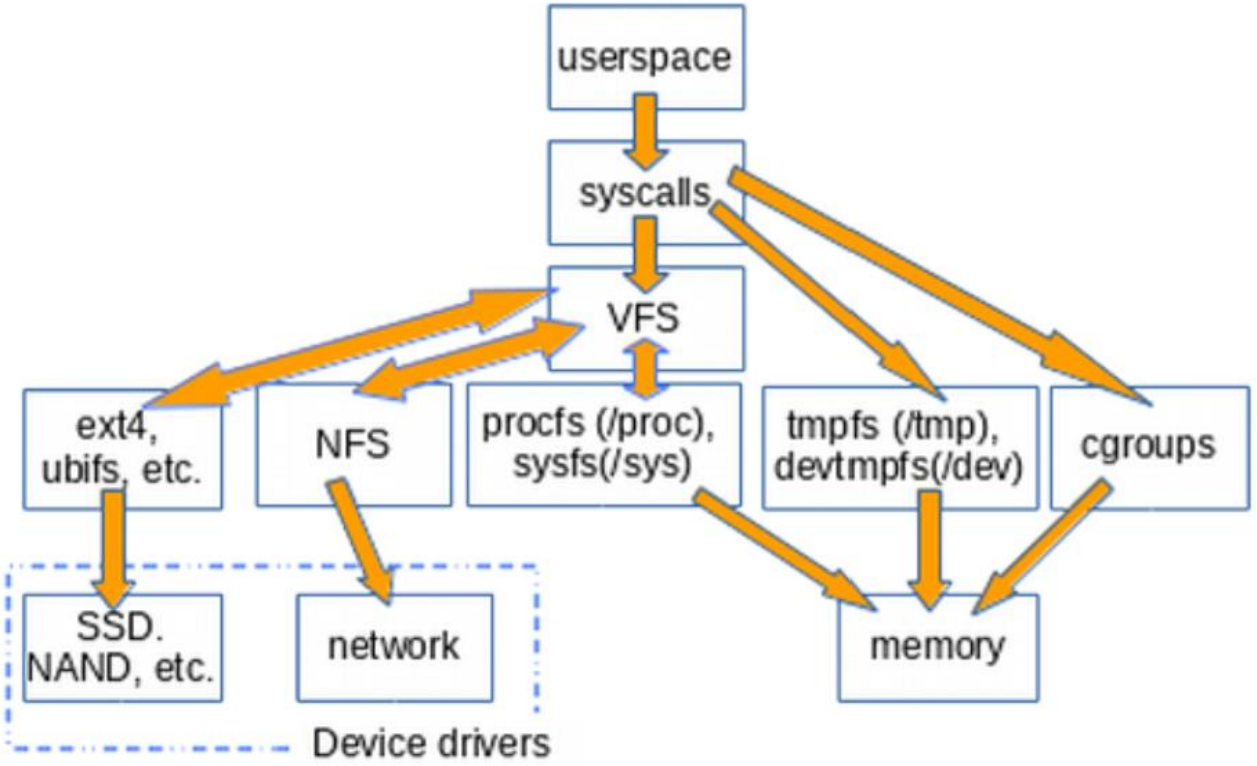
```
root@a24671bde3dc:~# file /dev/console
/dev/console: character special (136/0)
root@a24671bde3dc:~# ls -la /dev/console
crw--w---- 1 root tty 136, 0 May 29 18:47 /dev/console
root@a24671bde3dc:~#
```

```
root@a24671bde3dc:~# file
bash: file: command not found
```

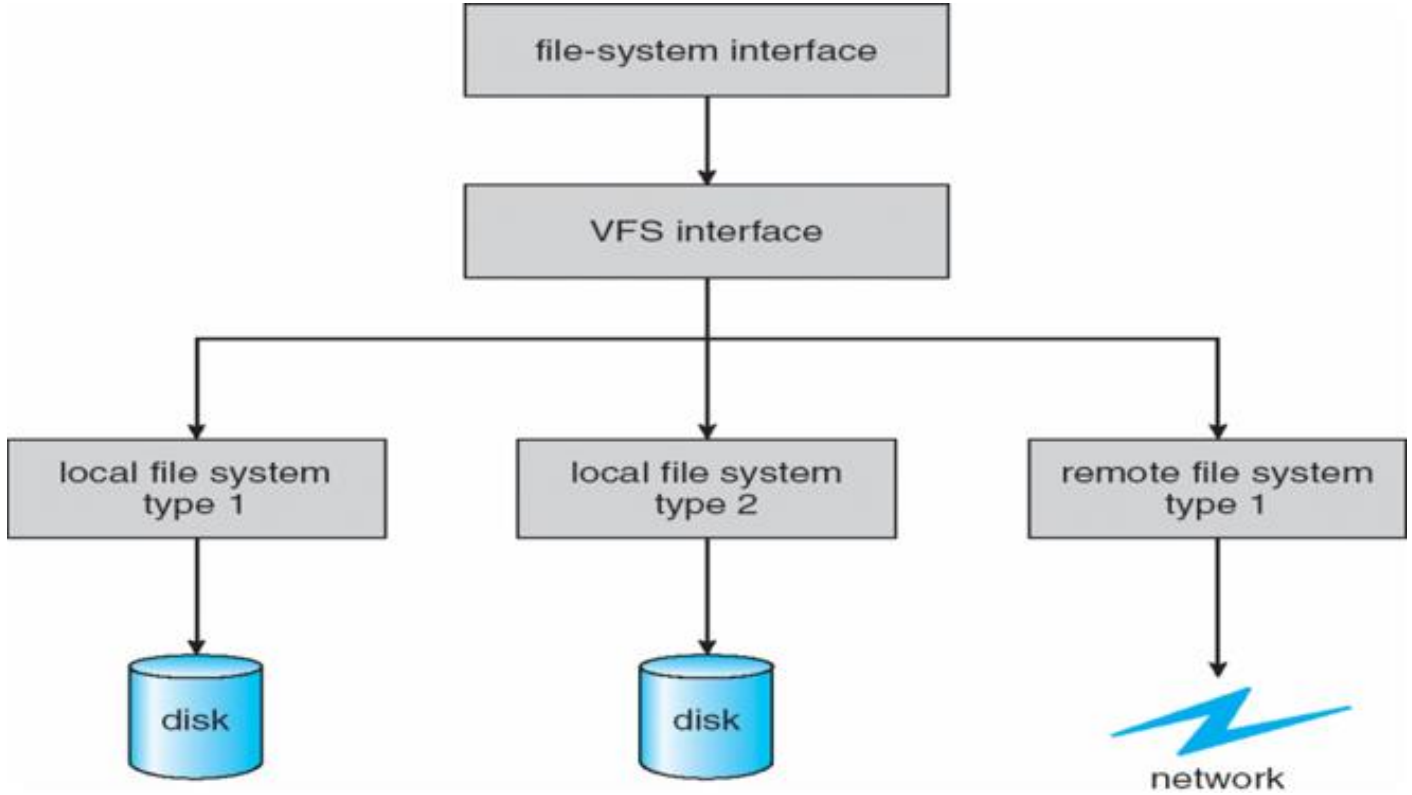
```
root@a24671bde3dc:~# apt install coreutils
```

/dev/console是Linux下的虚拟终端设备（tty），但是被统一视为文件（character special）

Linux Virtual File System Diagram



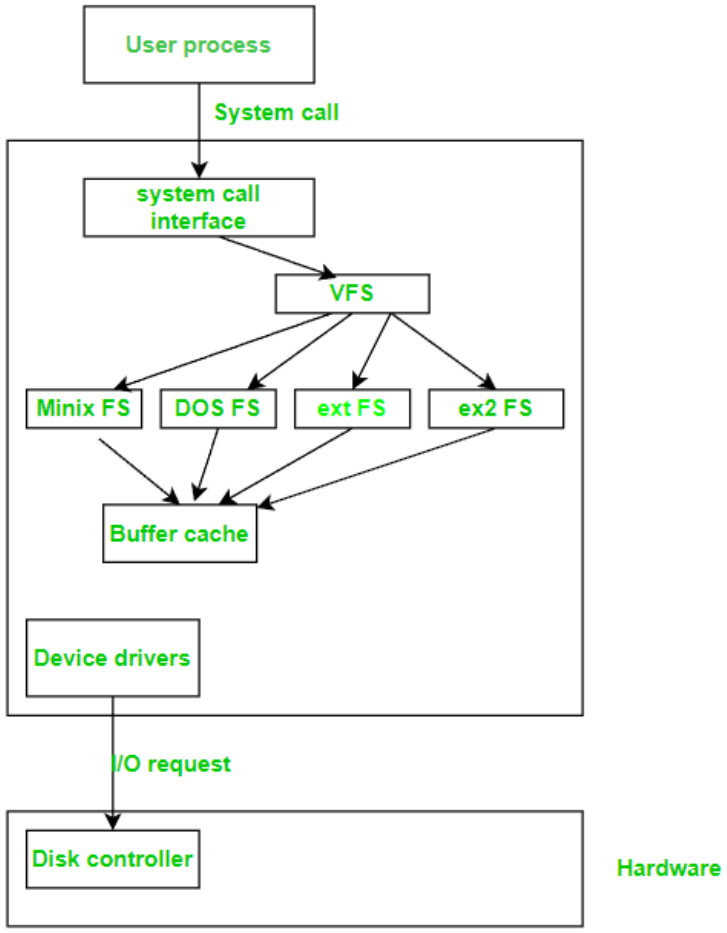
Virtual File System



The Virtual File System (also known as the Virtual Filesystem Switch) is the software layer in the kernel that provides the filesystem interface to userspace programs. It also provides an abstraction within the kernel which allows different filesystem implementations to coexist.

虚拟文件系统

Virtual File System

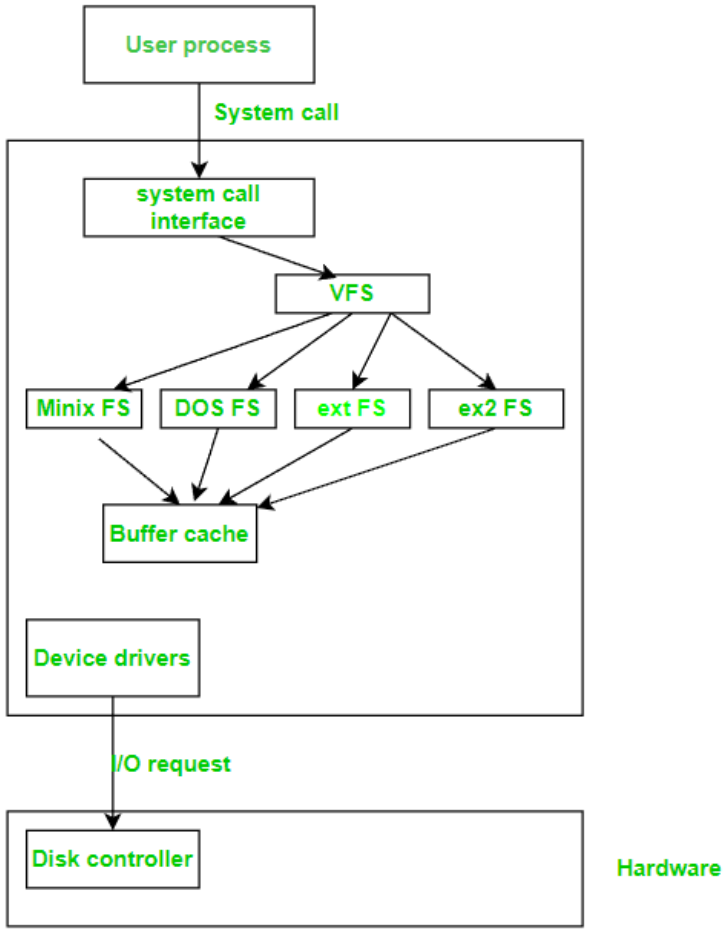


The user process initiates the execution of the system call with the help of the VFS method.
Linux中，用户进程发起的文件相关系统调用，都是发到VFS层

VFS interprets this stimulus into an appropriate internal file system call which in turn activates the concerned mapping function of the specified file system.
VFS会把系统调用请求翻译为具体文件系统的文件操作

虚拟文件系统

Virtual File System



The user initiates a file-oriented system call.
1.Kernel calls function in VFS.
2.Execution of file system independent manipulation and initiation of target file system call（执行VFS中的操作，为目标/物理文件系统的调用进行初始化）.
3.mapping of calls from VFS to the target file system.
4.The target file system converts the request into device driver execution（物理文件系统会将上层系统调用请求转给设备驱动程序）.
5.Appropriate action is completed as per the guidance of the device driver function and the results are obtained（驱动程序驱动设备，完成I/O后返回结果）.

文件系统实现

File System Implementations

02

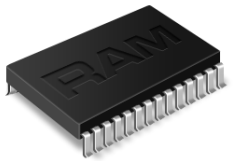
数据结构

文件系统实现中的核心问题:
分区、目录、文件等文件系统关键信息存放在哪里?

为了保证在关机断电之后，文件系统数据能够完好地保存，
文件系统相关的数据应在磁盘上保留一套完整的管理数据结构。

磁盘数据结构

启动控制块(Boot Control Block)
超级块（Superblock）
文件控制块（File Control Block）



内存数据结构

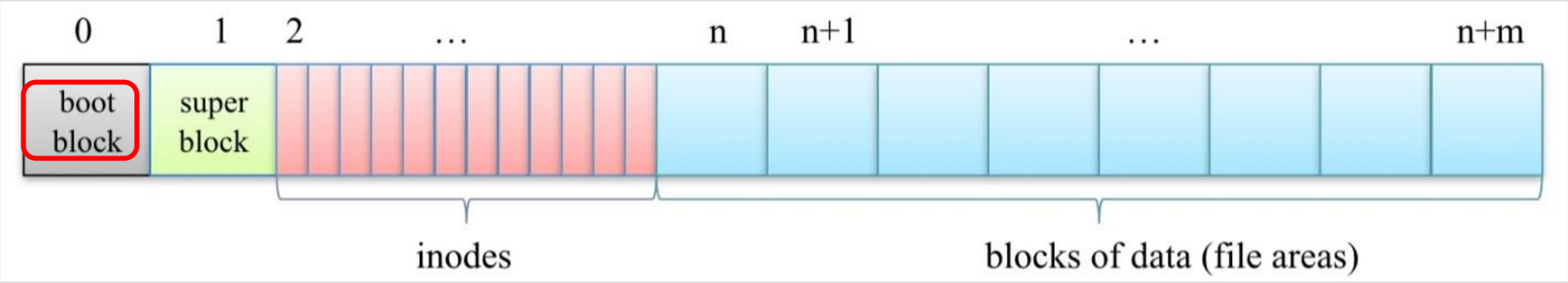
文件系统加载表
目录缓存
系统打开文件表
进程打开文件表

Q2： 为什么需要内存数据结构呢?

提速

磁盘数据结构

磁盘数据结构逻辑示意图

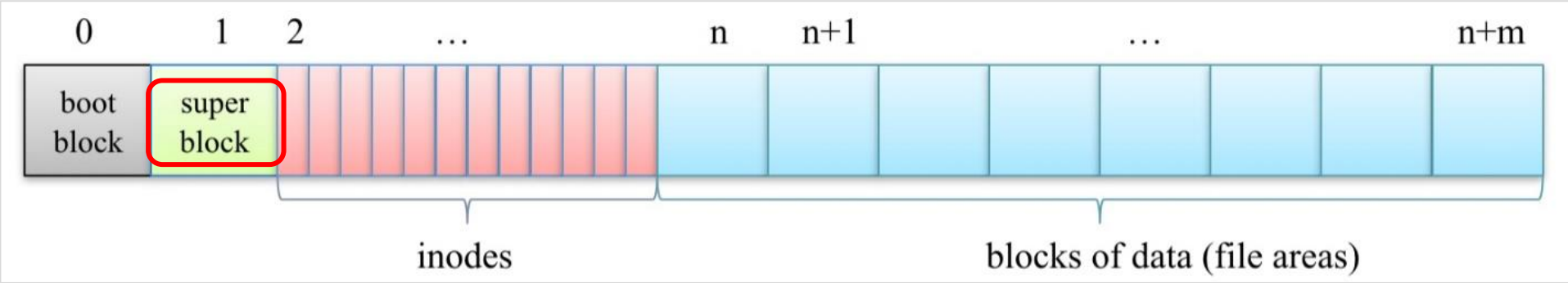


启动扇区(Boot block)

其中包括用于加载系统的初始代码

磁盘数据结构

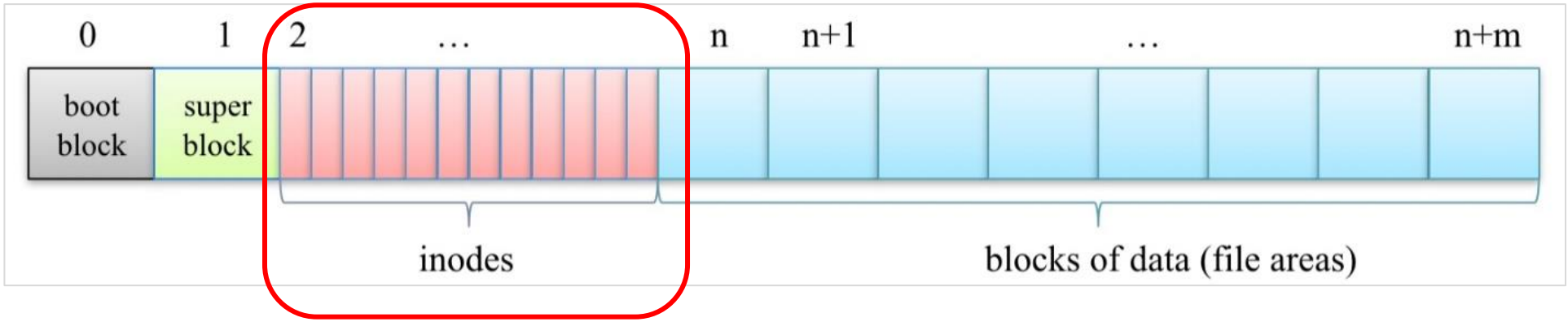
分区结构元信息（以Linux EXT文件系统为例）



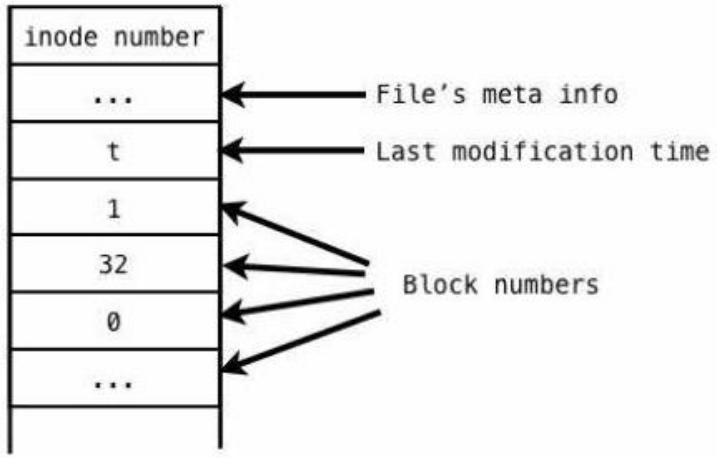
超级块（包含分区信息）

磁盘数据结构

目录与文件结构元信息（以Linux EXT文件系统为例）



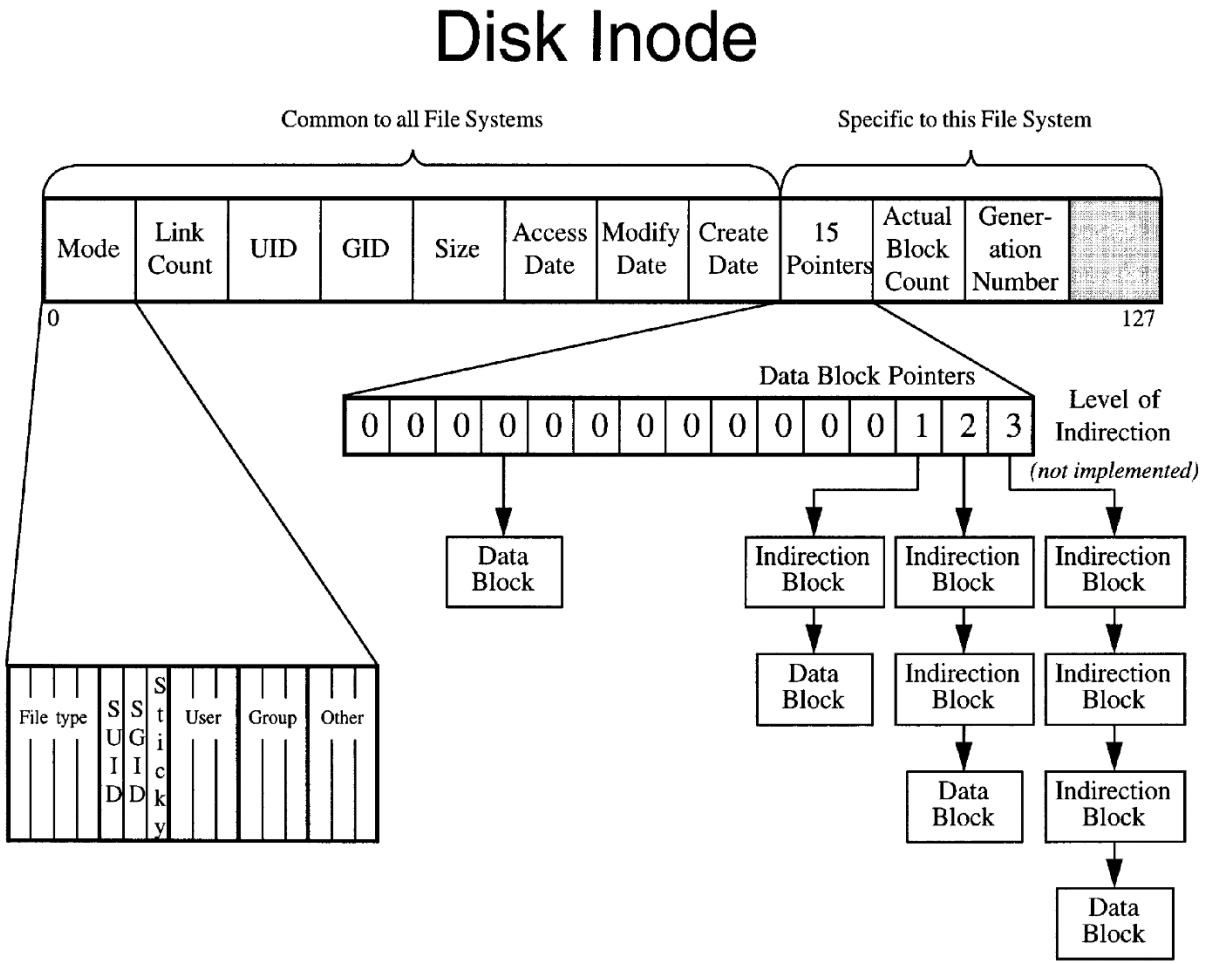
Linux系统下，将目录与文件统一对待（目录即文件）



磁盘数据结构

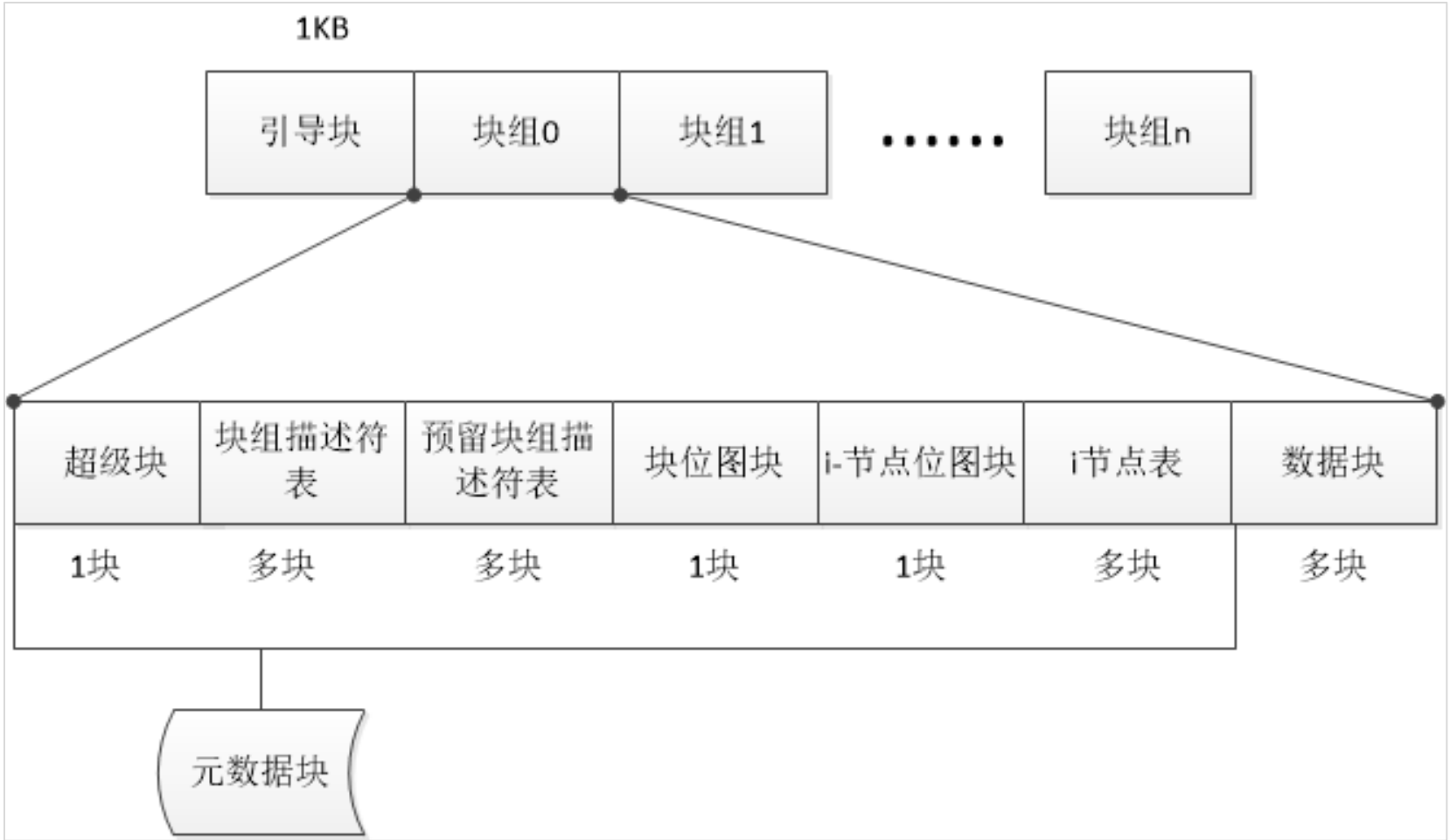
目录与文件结构元信息（以Linux EXT文件系统为例）

inode



磁盘数据结构

- **磁盘数据结构示例:Linux ext4**



内存数据结构

- 文件系统工作过程中需要内存数据结构的支持

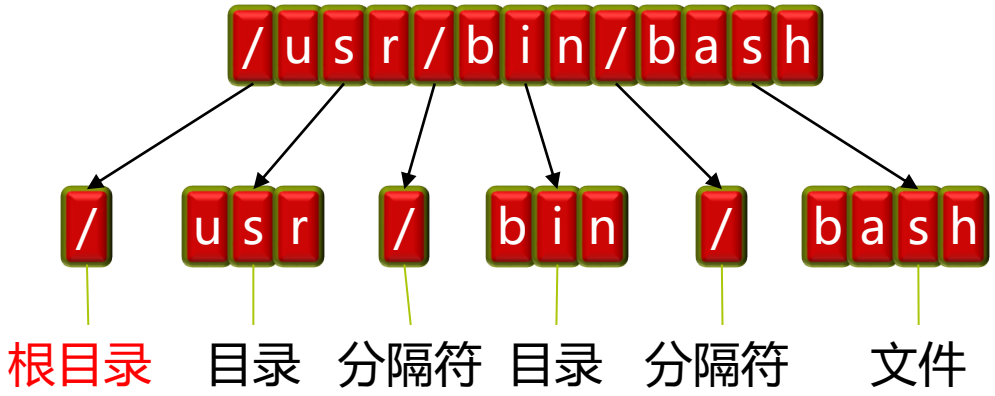
Why?

内存数据结构

- 文件系统工作过程中需要内存数据结构的支持

考虑几个问题：

给定一个绝对路径（或相对路径），如何定位到文件，并进行访问?

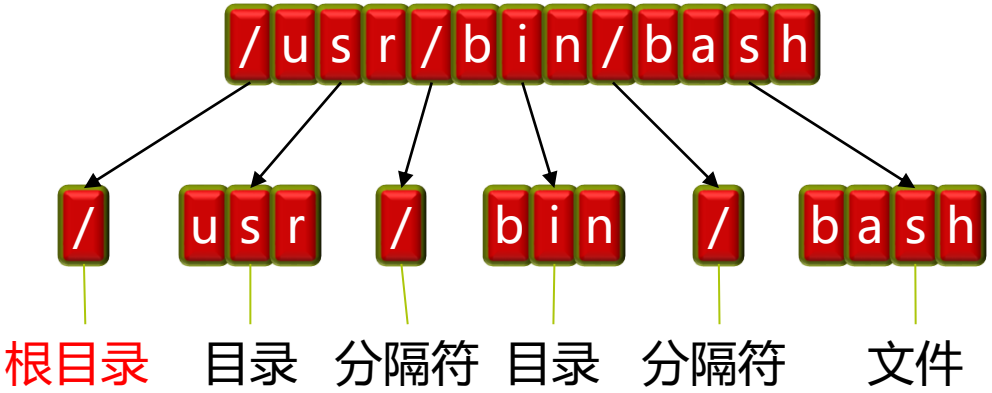


内存数据结构

- 文件系统工作过程中需要内存数据结构的支持

考虑几个问题：

给定一个绝对路径（或相对路径），如何定位到文件，并进行访问?



步骤1：在磁盘上找到根目录/对应的inode

Linux Ext4中，根目录对应的是分区上的第2个inode（第1个inode预留）

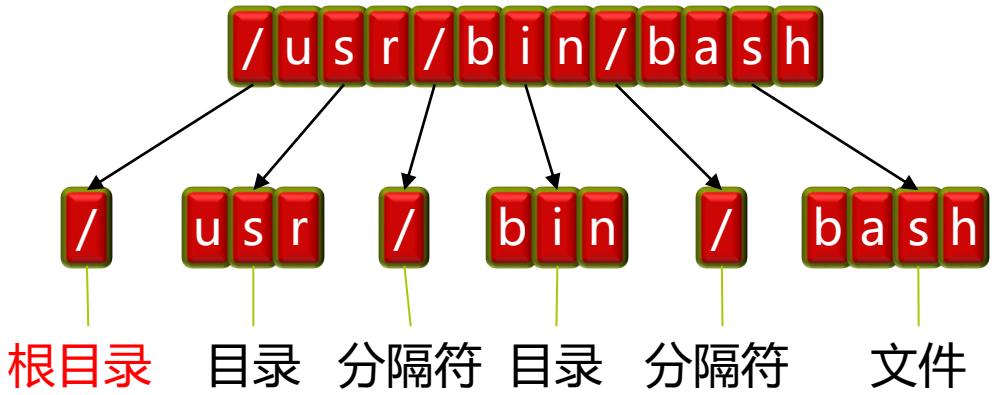
如何定位到第2个inode呢？ 需要用到superblock中的块组设置信息，块组0的描述信息

内存数据结构

- 文件系统工作过程中需要内存数据结构的支持

考虑1个问题：

给定一个绝对路径（或相对路径），如何定位到文件，并进行访问?



步骤1：在磁盘上找到根目录对应的inode

步骤2：在根目录的内容中查找usr对应的目录项

步骤3：从/usr目录中查找bin对应的目录项

步骤4：从/usr/bin目录中查找bash对应的目录项

步骤5：从找到的目录项中定位到bash文件的inode

内存数据结构

- 需要一级一级解析路径
  - 每一级路径，从磁盘取

具体如何读取磁盘内容，与文件物理结构有关： 下讲讨论

但不管如何，每一级目录内容都需要从磁盘中读取

其中的问题：磁盘I/O速度比内存访问速度要慢好几个数量级，怎么处理?

解决思路：利用内存，为频繁访问的目录或文件结构建立缓存。

内存数据结构

- 为了提升文件系统效率，操作系统会准备多种内存数据结构（用于缓存关键和常用的文件系统数据）
  - 文件系统加载表（In-Memory Mount Table）
  - 目录缓存（In-Memory Directory Structure Cache）
  - 全系统打开文件表
  - 进程打开文件表

内存数据结构

FS内存数据结构1：文件系统加载表（File System Mount Table）

例：Linux有一个系统配置文件 /etc/fstab (指定OS启动时自动加载的文件系统列表)

```
r /etc/fstab
# Created by anaconda on Sun Jan 15 16:37:50 2017
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=ca726e5d-df3c-40b7-8912-1d4089c09454 /                       ext4    defaul
ts        1 1
UUID=3e351356-baba-4aa7-9602-49688f64265id swap                    swap    defau
lts        0 0
/mnt/swapfile swap   swap  defaults  0  0
UUID=b106ba7f-59bc-412e-8217-45b8b0c11884  /home  ext4  defaults  0  2
~
~
~
~
~
~
~
~
~
~
~
~
"/etc/fstab" 10L, 518C
```

内存数据结构

FS内存数据结构1：文件系统加载表（File System Mount Table）

将分区控制块（e.g. Linux Ext文件系统的super block）中的文件系统元信息读入内存

内存数据结构

FS内存数据结构2：目录项缓存

将常用目录项缓存在内存中，加速路径解析效率

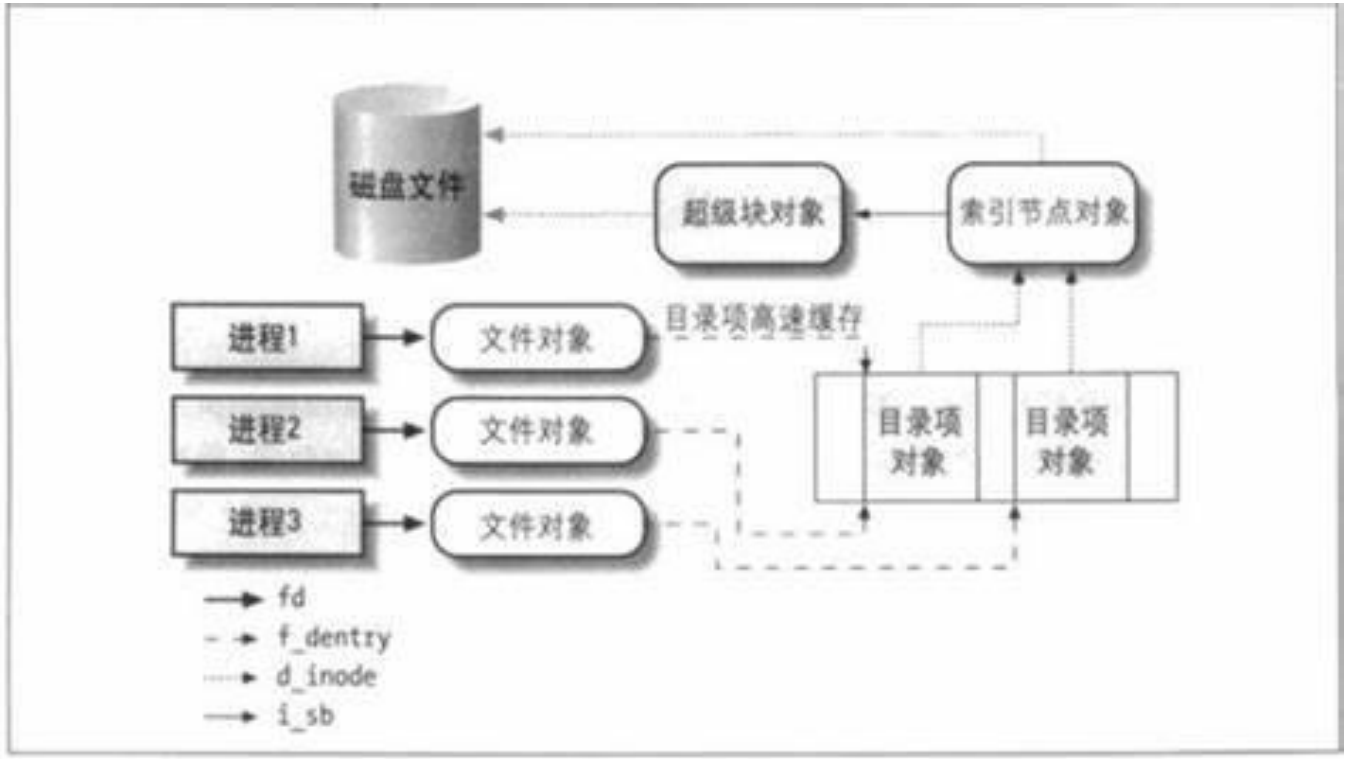


图 12-2：进程与 VFS 对象之间的交互

内存数据结构

FS内存数据结构3：系统打开文件表和进程打开文件表

每个进程打开的文件相关信息，会被记录在内存中

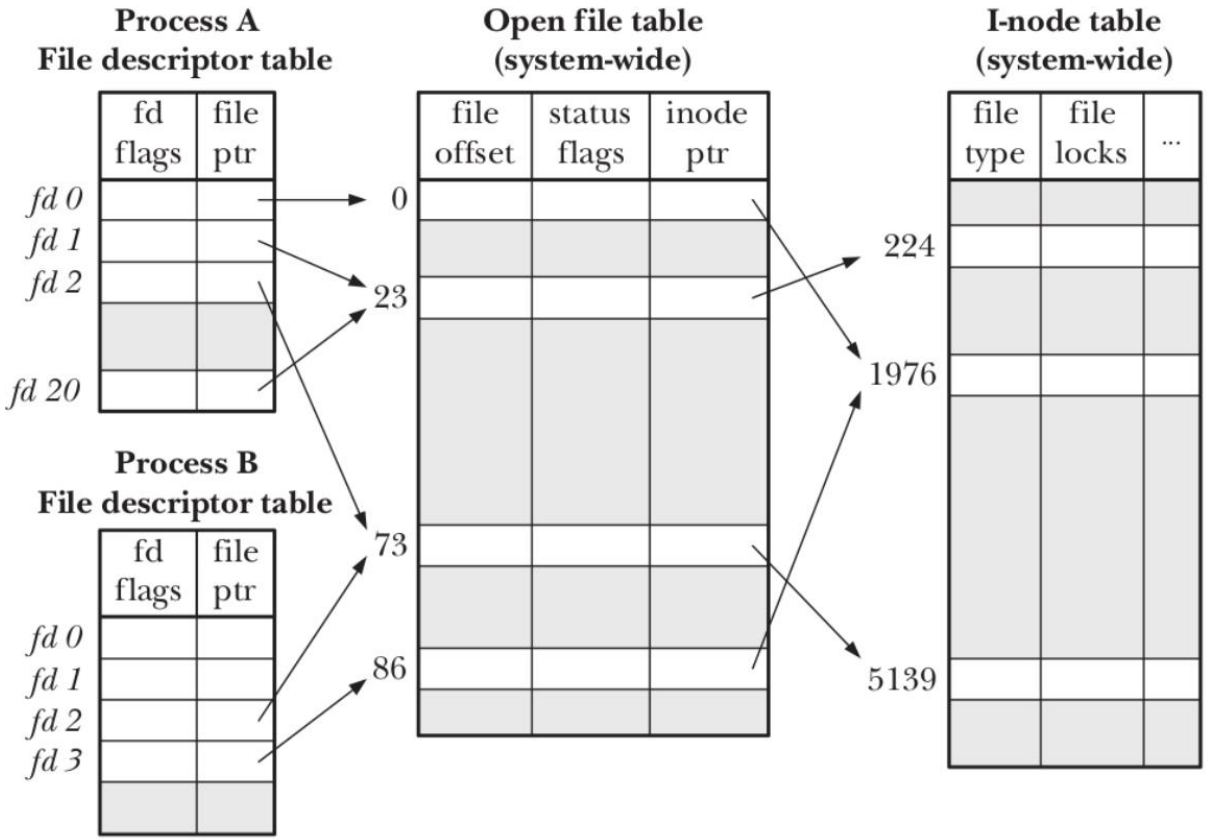


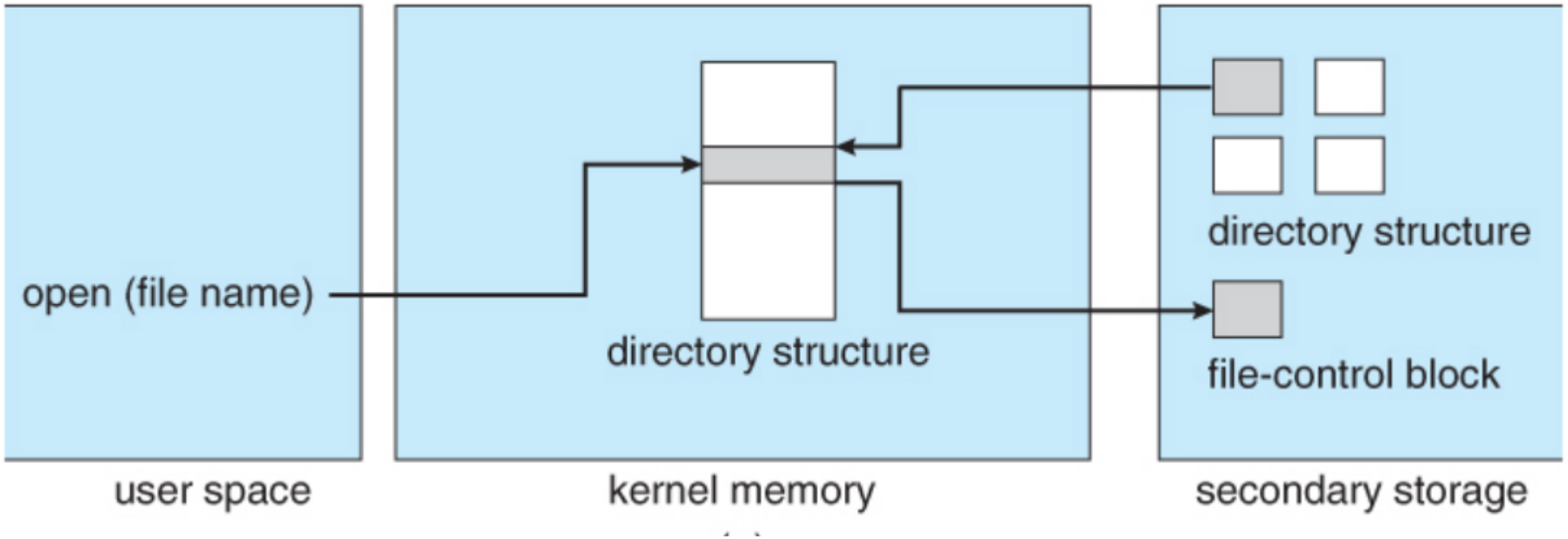
Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

文件创建操作

1. Create FCB for the new file
2. Update directory contents
3. Write new directory contents to disk (and may cache it as well)

文件操作实现

文件打开操作



- When a file is accessed during a program, the open( ) system call reads in the FCB information from disk, and stores it in the system-wide open file table.
- An entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned by the open( ) system call. UNIX refers to this index as a ***file descriptor***, and Windows refers to it as a ***file handle***.

文件操作实现

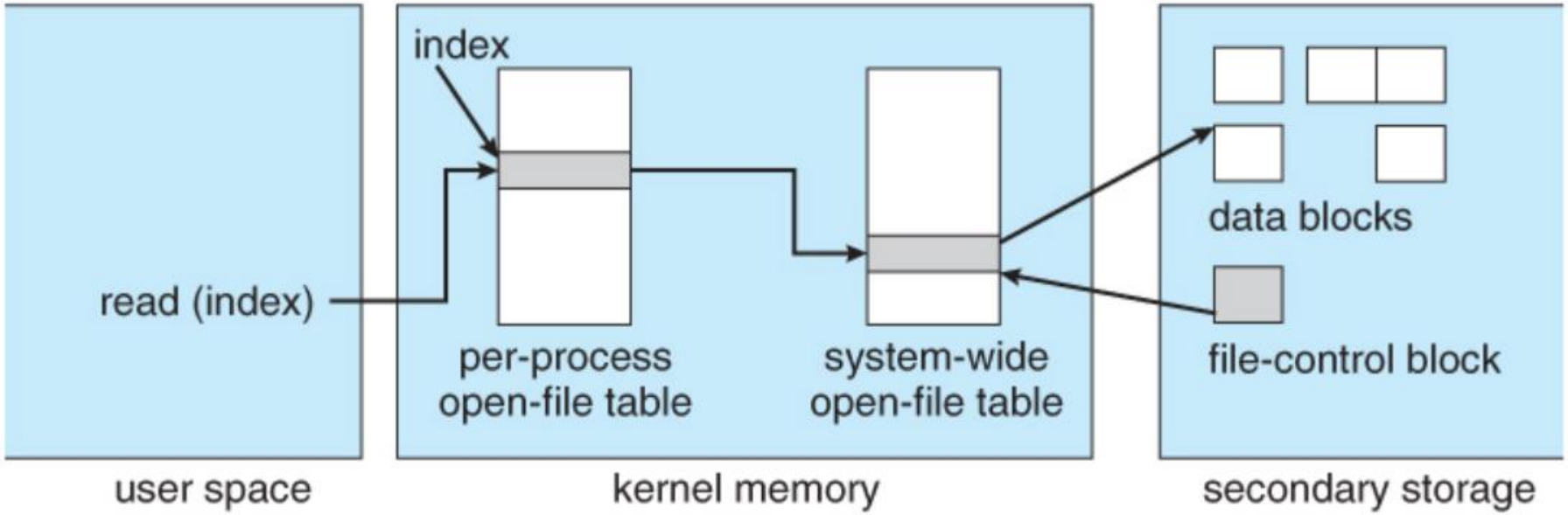
Linux的open系统调用

```
int fd = open(FILEPATH, W_RDONLY);
```

用户不需要知道内核中VFS和具体的文件系统实现，就可以通过文件名文件路径，自如地进行文件读写，获取文件信息，修改文件属性。

文件操作实现

文件读操作



文件关闭操作

1. Per process open table entry is removed
2. System wide open table reference count decremented by 1
   - If this value becomes 0, then updates copied back to disk (if needed)
   - Remove system wide open table entry

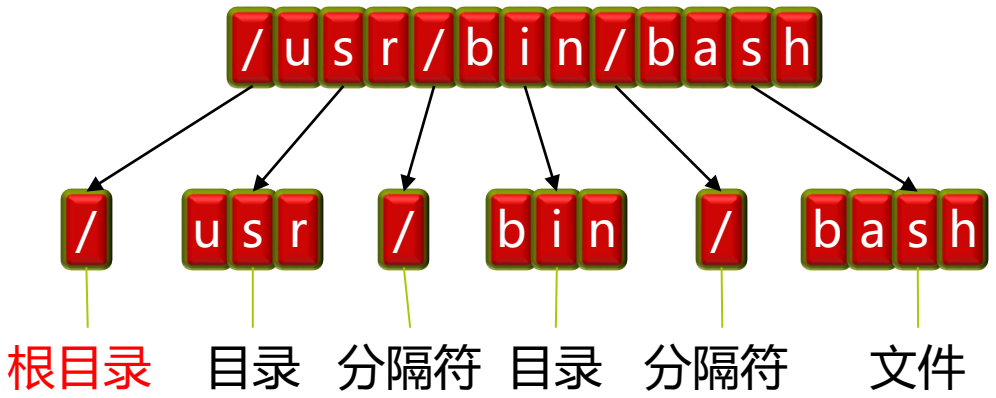
目录实现

Implementation of Directory

03

目录与文件按名存取

- **文件路径名解析**
  - 文件路径名解析会在文件open操作中被使用，很重要
  - 路径名解析的目的：根据文件路径，找到对应文件控制块
  - 做法：解构路径名，将其拆分成一级一级的目录名及最后一级的文件名



目录与文件按名存取

- **文件路径名解析**

绝对路径

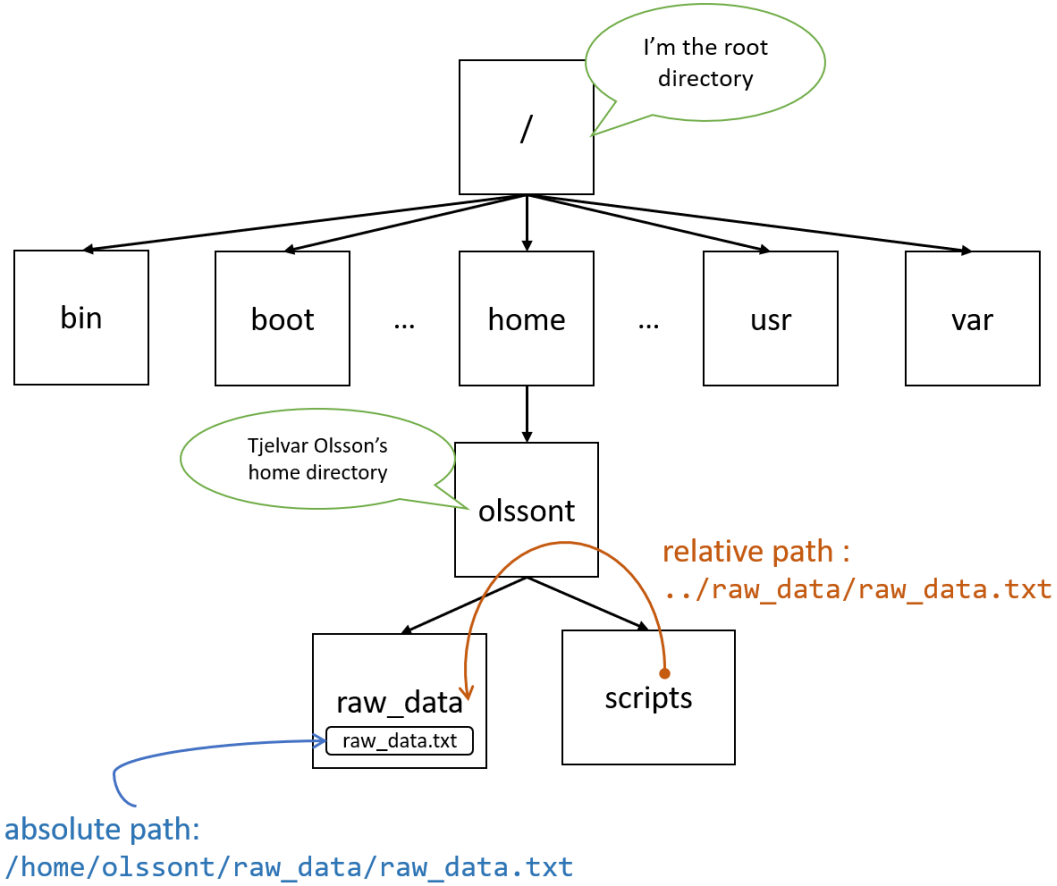
If the pathname starts with the ‘/’ character, the starting lookup directory is the root directory of the current process.

解析的起点：current->fs->root

相对路径

If the pathname does not start with the ’ /’ character, the starting lookup directory of the resolution process is the current working directory of the process.

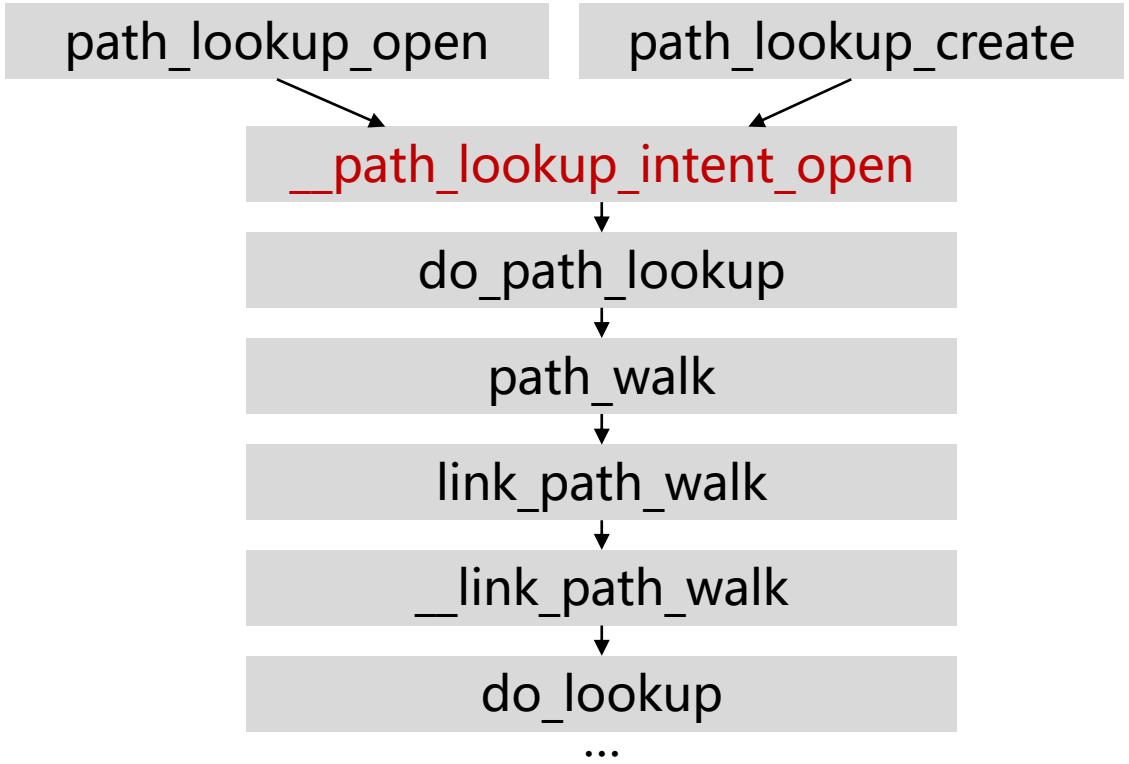
解析的起点：current->fs->pwd



目录与文件按名存取

• 文件路径名解析

- Linux中解析代码的函数调用序列



不管是path_lookup_open()还是path_lookup_create()，最终都是调用__path_lookup_intent_open()来实现查找文件的功能。

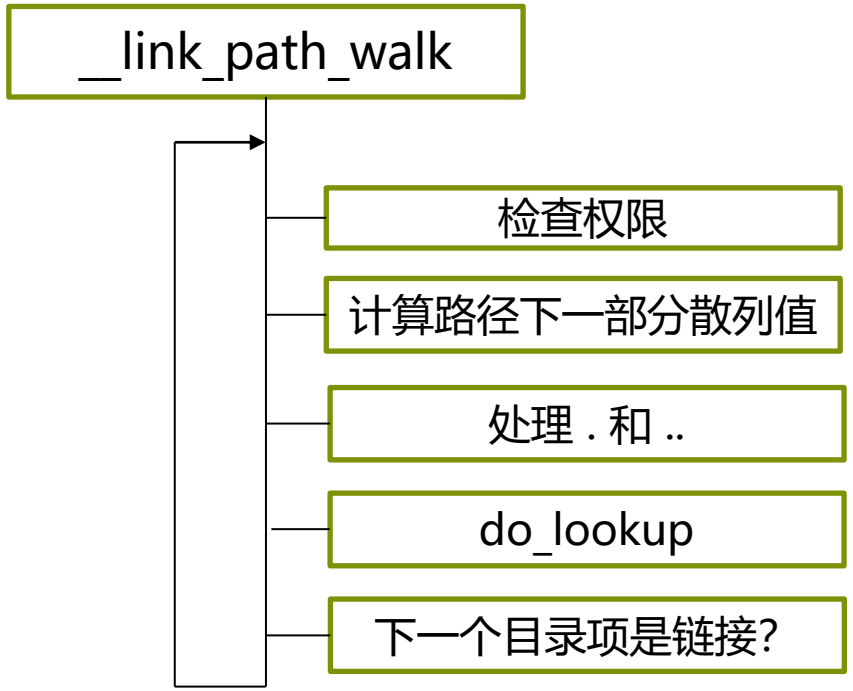
- 查找时，会遍历路径的过程中，会逐层地将各个路径组成部分解析成目录项
- 如果此目录项在目录项缓存中，则直接从缓存中获取；如果该目录项不在目录项缓存，则进行一次实际的读盘操作，从磁盘中读取该目录项，并得到目录项所对应的索引节点
- 得到索引节点之后，建立索引节点与该目录项的联系

如此循环，直到找到目标文件对应的目录项，也就找到了文件的索引节点，而由索引节点找到对应的超级块对象，就可知道该文件所在的文件系统的类型。知道了文件系统的类型，VFS就能调用这个文件系统相对应的索引节点操作函数集来进行操作。

从磁盘中读取该目录项对应的索引节点；将引发VFS和实际的文件系统的一次交互。

目录与文件按名存取

- **Linux文件路径解析实现：__link_path_walk()**



如果是链接文件则进行特殊的处理do_follow_link

目录实现结构

- **为了路径的快速解析，需要为目录设计专门数据结构**

Option 1：Linear List

- the simplest and easiest directory structure to set up, but with some drawbacks
- Finding a file requires a linear search.
- Deletions can be done by moving all entries, flagging an entry as deleted, or by moving the last entry into the newly vacant position.
- Sorting the list makes searches faster, at the expense of more complex insertions and deletions.
- A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.
- More complex data structures, such as B-trees, could also be considered.

Option 2：Hash Table

- A hash table can also be used to speed up searches.
- Hash tables are generally implemented ***in addition to*** a linear or other structure

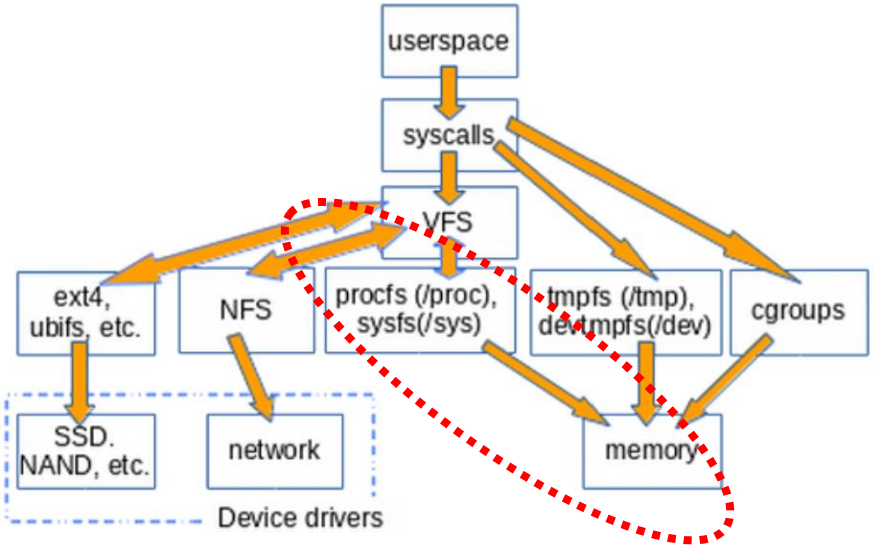
Summary

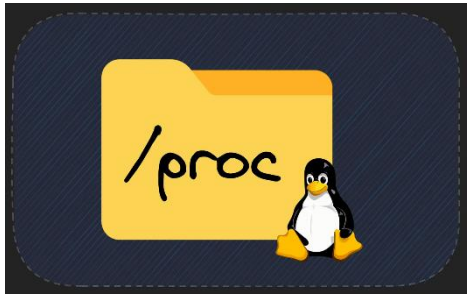
小结：

- 文件系统结构
- 文件系统实现要点
- 目录实现

Linux /proc

The /proc directory in Linux

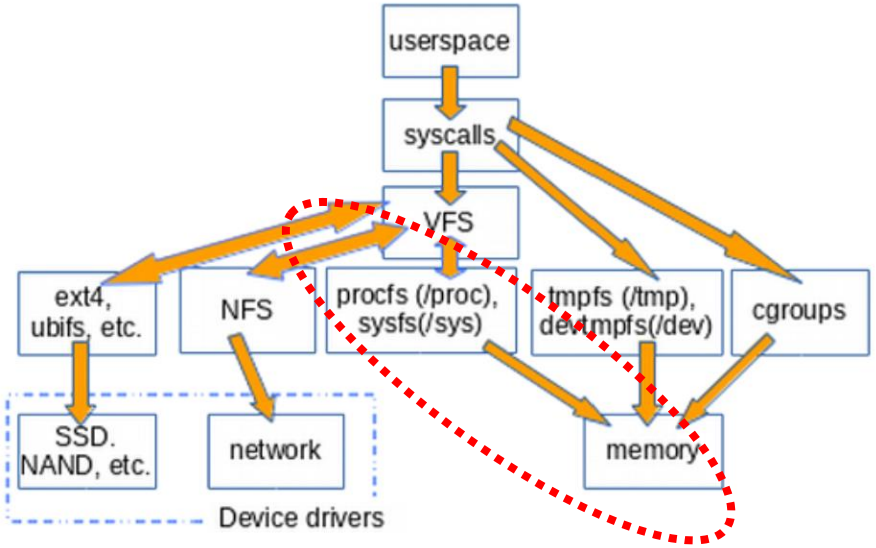


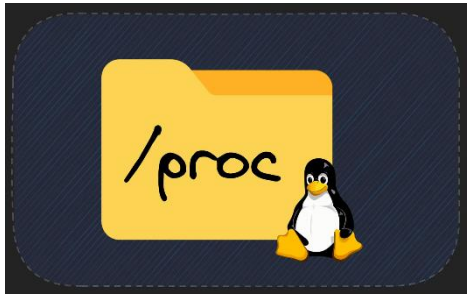



```
root@a24671bde3dc:/proc# ls -la
total 4
dr-xr-xr-x 308 root root     0 May 29 18:36 .
drwxr-xr-x   1 root root  4096 May 29 18:36 ..
dr-xr-xr-x   9 root root     0 May 29 18:36 1
dr-xr-xr-x   9 root root     0 May 29 19:23 148
drwxrwxrwt   2 root root    40 May 29 18:36 acpi
-r--r--r--   1 root root     0 May 29 19:02 buddyinfo
dr-xr-xr-x   4 root root     0 May 29 18:36 bus
-r--r--r--   1 root root     0 May 29 19:02 cgroups
-r--r--r--   1 root root     0 May 29 19:02 cmdline
-r--r--r--   1 root root 22541 May 29 19:02 config.gz
-r--r--r--   1 root root     0 May 29 19:02 consoles
-r--r--r--   1 root root     0 May 29 19:02 cpuinfo
-r--r--r--   1 root root     0 May 29 19:02 crypto
-r--r--r--   1 root root     0 May 29 19:02 devices
-r--r--r--   1 root root     0 May 29 19:02 diskstats
-r--r--r--   1 root root     0 May 29 19:02 dma
dr-xr-xr-x   4 root root     0 May 29 19:02 driver
-r--r--r--   1 root root     0 May 29 19:02 execdomains
-r--r--r--   1 root root     0 May 29 18:36 filesystems
dr-xr-xr-x  11 root root     0 May 29 18:36 fs
-r--r--r--   1 root root     0 May 29 19:02 interrupts
```

Linux /proc

The /proc directory in Linux





/proc/sysvipc

```
root@a24671bde3dc:/proc/sysvipc# ls
msg  sem  shm
root@a24671bde3dc:/proc/sysvipc# cat shm
       key      shmid perms                  size  cpid  lpid nattch   uid   gid  cuid  cgid      atime      dtime
ctime                  rss                  swap
```

外存上存放的数据（ ）。

A. CPU可直接访问

B. CPU不可访问

C. 是高速缓存中的信息

D. 必须先装入内存才可以被CPU访问

操作系统内核对磁盘上的文件是以（ ）为单位进行读写。

A. 块

B. 记录

C. 区段

D. 页面

在Unix系统中，某文件的使用权限设置为754，则表示（ ）。

A. 文件主可读、写、执行

B. 同组用户仅能读

C. 其他用户可读、写、执行

D. 同组用户仅能写

在文件系统的支持下，用户进行文件访问时需要知道文件存放的物理地址。

A. True

B. False

谢 谢!
Thank you!