目录

[目录 1](#_Toc405222818)

[第二题: 1](#_Toc405222819)

[第三题: 6](#_Toc405222820)

[第四题: 9](#_Toc405222821)

[第六题: 9](#_Toc405222822)

[第七题. 第八题: 9](#_Toc405222823)

## 第二题:

**1./dev/sdb7代表的含义是？它是主分区还是拓展分区？拓展分区有几个？**

->我将在后面把拓展分区称之为 扩展分区

->磁盘容量与主分区、扩展分区、逻辑分区的关系：

硬盘的容量＝主分区的容量＋扩展分区的容量

扩展分区的容量＝各个逻辑分区的容量之和

->一块物理硬盘只能有: 一到四个主分区(但其中只能有一个是活动的主分区),或一到三个主分区,和一个扩展分区

对于Linux/i386来说，分区1-4是主分区，5-15是逻辑分区。

Linux 中规定，每一个硬盘设备最多能有 4 个主分区（其中包含扩展分区）构成，任何一个扩展分区都要占用一个主分区号码，也就是在一个硬盘中，主分区和扩展分区一共最多是 4 个

->hd: IDE硬盘 sd: SCSI硬盘

答案： 1./dev/sdb7 代表的含义是 第二个(b代表2)SCSI硬盘的第7个分区

2.为扩展分区里面的逻辑分区

3.扩展分区只能有一个（也可以没有），其后的标号为5.6.7…等为逻辑分区，逻辑分区的容量总和为扩展分区的容量。

**2. 磁盘的第一个扇区存放的是什么？它的大小多大？如何划分的？**

->一般的扇区大小都为512字节，但现在的扇区也有可能是4KB的。第一批4096字节扇区磁盘将每个物理扇区翻译成8个512字节的逻辑扇区。对于BIOS，操作系统和所有的磁盘工具来说，磁盘扇区大小看来起仍然是512字节，虽然实际的物理扇区已经是4096字节了。

->计算机首先从最初一个扇区开始读取设备，然后检查这个扇区的最后两个字节，也就是第511和512字节。如果最后2个字节不是0x55和0xAA，计算机会认为此设备上无所需的启动程序，就会报一个不能启动的错误。如果计算机确认第一个扇区的最后两个字节正好是55 AA，那它就认为这个扇区的开头是启动程序，并开始执行这个程序

答案: 1.磁盘的第一个扇区存放的是：引导区信息，用于加载并转让处理器控制权给操作系统

2.大小为512字节

3. 硬盘的第一个扇区被称之为 Boot Sector，由 MBR (MasterBoot Record)、DPT (Disk Partition Table) 和 Boot Record ID 三部分组成

硬盘主引导扇区 = 硬盘主引导记录（MBR）+ [硬盘分区表](http://baike.baidu.com/view/1385.htm)（DPT）+ 结束标志

1. MBR 又称作主引导记录，占用 Boot Sector 的前 446 个字节(0 ~ 0x1BD)。存放系统主引导程序，负责从活动分区中装载并运行系统引导程序。

作用检查[分区表](http://baike.baidu.com/view/1412956.htm)是否正确以及确定哪个分区为[引导分区](http://baike.baidu.com/view/1328673.htm)，并在程序结束时把该分区的启动程序（也就是操作系统引导扇区）调入内存加以执行

2.DPT 即主分区表，占用 64 个字节 (0x1BE ~ 0x1FD)，记录了磁盘的基本分区信息。主分区表分为四个分区项，每项 16 字节，分别记录了每个主分区的信息 (因此最多可以有 4 个主分区)。

3.Boot Record ID 即引导区标记，占用两个字节 (0x1FE ~ 0x1FF)。对于合法引导区，它等于 0xAA55，这是判别引导区是否合法的标志

**3.如何安装有windows和linux双系统，MBR如何使用？又如何区分两个系统？**

->开机默认用MBR引导：(安装有Windows7系统上的机器，安装用Ubuntu后，开机会自动用grub2引导)

进入相应硬盘执行： Mbrfix.exe /drive 0 fixmbr

这里的drive 0, 是指定要修复的硬盘，编号可以从“磁盘管理”看到，只有一块儿硬盘的一般是drive 0。然后确认，即可。

Mbrfix的下载地址：/Files/sunjie21/mbrfix.zip

如果是在windows系统下安装的linux，那么安装好之后会由grub2来引导启动，可将其改为用MBR引导（方法如上…）

->FDISK／MBR  
参数功能：重新建立主磁盘的主引导记录（Master Boot Record）。卸载WindowsNT或Windows2000后消除在主引导记录上记载的系统启动选择；当有病毒感染主引导记录时，用来清除病毒相当有效。

->单硬盘的系统共存：一般有两种方法

1.修改主引导记录，即MBR，将最后的JMP指令跳到自己的代码上来

2.修改主分区第一个扇区的引导代码，以实现多系统的共存

-> MBR不随操作系统的不同而不同，意即不同的操作系统可能会存在相同的MBR，即使不同，MBR也不会夹带操作系统的性质。具有公共引导的特性

->特殊形式的MBR会占用多个扇区，如 Lvyanan 的1JF9占用2个扇区，Pauly 的 XORLDR 占用30来个扇区， GRUB 的MBR占用18个扇区，FBINST 占用64个扇区，PloP Boot Manage 占用62个扇区（总共63个扇区）

答案： 1.安装windows和linux的双系统可以有很多种方法，可以在windows环境下安装linux系统，也可以在linux环境下安装windows系统。使用的可以使镜像文件从硬盘安装，也可以从u盘启动，也可以使用光盘。

我们比较常使用的是在windows环境下安装linux系统，并在没有正版光盘的条件下采用u盘启动安装（或者刻录一张光盘）。

一：为linux准备分区，文件类型暂设为FAT32

二：如果没有制作linux系统盘，就在windows系统下制作一个系统盘

三：用U盘启动，安装linux

四：用grub引导(一般从win7安装完linux都会是默认grub引导)

2.我不是很明白 “MBR如何使用” 到底是要表达什么意思，按照我自己的理解把它认为是MBR如何工作的。

主引导记录 MBR的作用：1.就是检查分区表是否正确 2.确定哪个分区为引导分区，并在程序结束时把该分区的启动程序(也就是操作系统引导扇区)调入内存加以执行。(主引导程序将控制权交给活动分区的引导记录由引导记录加载操作系统)

MBR最后有一个JMP指令将直接跳转到操作系统引导扇区。

看了很久也实在没有很清楚理解这其中的原理，现在按照我自己的理解来解释MBR：首先在硬盘的第一个扇区一定会有一个引导扇区，这个扇区在BIOS的准备工作完成后跳转到这里进行，这时的MBR实现其自己的作用（如上），它会根据主分区表寻找可启动的操作系统的位置 （MBR启动时只是去检测80和起始扇区号，对分区信息不做完整检测），之后跳转到可执行的操作系统处加载程序，将控制权交给操作系统引导

3.区分两个操作系统（个人理解）：安装两个操作系统可以扩展主分区表内容， 当系统主引导记录查询到主分区表时，发现有两个可启动的操作系统，这时的选择 权交给用户选择需要启动的操作系统（可以设置默认启动的操作系统，如果启动的操作系统是C盘中的，则无需再次执行MBR），然后跳转到 分区执行该分区的MBR，通过该分区的MBR加载要执行的操作系统，之后正常启动

**4. 设备文件里主要存放什么信息？如果设备插入以后，却没有相应的设备文件，可能问题**

**是什么？**

答案： 1./dev这个目录中包含了所有Linux系统中使用的外部设备。但是这里并不是放的外部设备的驱动程序，它实际上是一个访问这些外部设备的端口

2. ->有可能是硬件的问题 ->没有安装驱动程序

**5. /dev/null, /dev/zero是什么？他们一般有什么用处？**

->/dev/null

一、禁止标准输出.

eg:

    cat $filename >/dev/null

    # 文件内容丢失，不会输出到标准输出，.

二、禁止标准错误

eg:

    rm $badname 2>/dev/null

    #删除文件错误时，不会再有提示到终端，都丢到/dev/null里去了

三、禁止标准输出和标准错误的输出.

eg1:

    cat $filename 2>/dev/null >/dev/null

    # 如果"$filename"不存在，将不会有任何错误信息提示.

    # 如果"$filename"存在, 文件的内容不会打印到标准输出.

    # 因此, 上面的代码根本不会输出任何信息.

    # 当只想测试命令的退出码而不想有任何输出时非常有用。

eg2:

    #-----------测试命令的退出 begin ----------------------#

    ls dddd 2>/dev/null 8

    echo $?    //输出命令退出代码：0为命令正常执行，1-255为有出错。

    #-----------测试命令的退出 end-----------#

    cat $filename &>/dev/null

四、清除日志文件内容

eg:

    cat /dev/null > /var/log/messages

    #  : > /var/log/messages   有同样的效果, 但不会产生新的进程.（因为:是内建的）

    cat /dev/null > /var/log/wtmp

五、 隐藏cookie而不再使用

eg:

    if [ -f ~/.netscape/cookies ]  # 如果存在则删除.

    then

        rm -f ~/.netscape/cookies

    fi

    ln -s /dev/null ~/.netscape/cookies

# 现在所有的cookies都会丢弃而不会保存在磁盘上了.

答案： 1./dev/null: 空设备。任何写入都将被直接丢弃，任何读取都将得到EOF。(无限设置接收数据，形象地称之为黑洞)

用途(具体使用如上例)：一、禁止标准输出 二、禁止标准错误 三、禁止标准输出和标准错误的输出. 四、清除日志文件内容 五、 隐藏cookie而不再使用

2. /dev/zero 零字节源，只能读取到无限多的零字节。

用途：->用来创建一个指定长度用于初始化的空文件，就像临时交换文件. ->创建ramdisk(保存一个较大的数据集在ramdisk, 比如一张表或字典,这样可以加

速数据查询, 因为在内存里查找比在磁盘里查找快得多.)

## 第三题:

1. **FAT文件系统碎片如何产生的？Ext2文件系统需要定期执行碎片整理吗？**

->数据单元：FAT和NTFS文件系统中称为“簇” ExtX中称为“块”

一个数据单元由若干个连续的扇区组成，大小总是2的整数次幂个扇区，存数据是，总是为其分配整数个数据单元大小的空间。

->文件分配存储单元时的分配策略（三种）：

一、第一可用分配策略，就是为一个文件分配了一个存储单元后还要继续为其分配时，操作系统会重新从文件系统的起始处搜索可用空间，这种分配方法很容易产生碎片，因为文件不是被作为整体一次性分配连续存储空间的。

二、为文件分配了一个存储单元后并不再回到起始处重新寻找可用空间，而是直接向后搜索

三、最佳分配策略、 即在为文件分配空间时，尽可能找到足够的连续空间以避免其碎片产生，但这种分配策略需要事先知道文件所需要的空间大小，而且如果以后文件增大需要增加分配单元时，新分配的单元可能处于其他位置而使文件产生碎片

->索引式文件系统(indexed allocation): 文件系统先格式化出 inode 与 block 的区块，假设某一个文件的属性与权限数据是放置到 inode 4 号，而这个 inode 记录了文件数据的实际放置点为 2, 7, 13, 15 这四个 block 号码，此时我们的操作系统就能够据此来排列磁盘的阅读顺序，可以一口气将四个 block 内容读出来！

->。FAT 这种格式的文件系统并没有 inode 存在，所以 FAT 没有办法将这个文件的所有 block 在一开始就读取出来。每个 block 号码都记录在前一个 block 当中.

答案： 1.FAT文件主要是文件不在连续的扇区，并且簇链表也损坏的情况下产生系统碎片

碎片整理的原因：文件写入的 block 太过于离散了，此时文件读取的效能将会变的很差（磁盘转好多圈…）。 这个时候可以透过碎片整理将同一个文件所属的 blocks 汇整在一起，这样数据的读取会比较容易

2.  Ext2 是索引式文件系统，基本上不太需要常常进行碎片整理的。但是如果文件系统使用太久， 常常删除/编辑/新增文件时，那么还是可能会造成文件数据太过于离散的问题，此时或许会需要进行重整一下的。（在常规的情况下是不太需要整理的）

1. **一个文件的大小小于一个block的大小，是否能够装入下一个文件内容？**

答案： 文件小于 block ，则该 block 的剩余容量就不能够再被使用了(磁盘空间会浪费)，每个 block 仅能容纳一个文件的数据

**3. 如果系统使用2KB的block，而文件系统有10000个小文件，每个文件大小是80B，那么**

**磁盘浪费多少空间？有什么解决方案？**

答案： 1.浪费的空间 （2048-80）\*10000（byte）

2. block 大小有 1K, 2K 及 4K 三种

如果我们整体文件中并没有单一文件需要特别大的内存空间的时候，我们可以将block设为1KB的，以减少资源的浪费，但是如果需要读写的文件整体上偏向大文件方向，那么我们就只能浪费一些空间了，不然inode压力增大可能导致 文件系统不良的读写效能

所以具体我们要根据不同的情况来考虑

1. **在硬盘上读取文件/etc/passwd，它的inode和block访问顺序是？**

->当我们在 Linux 下的 ext2 文件系统创建一个目录时， ext2 会分配一个 inode 与至少一块 block 给该目录。其中，inode 记录该目录的相关权限与属性，并可记录分配到的那块 block 号码； 而 block 则是记录在这个目录下的文件名与该文件名占用的 inode 号码数据

-> inode 本身并不记录文件名，文件名的记录是在目录的 block 当中

->系统通过挂载信息可以找到挂载点的 inode 号码(通常一个 filesystem 的最顶层 inode 号码会由 2 号开始)，此时就能够得到根目录的 inode 内容，并依据该 inode 读取根目录的 block 内的文件名数据，再一层一层的往下读到正确的档名。

-> 1.透过挂载点的信息找到 /dev/hdc2 的 inode 号码为 2 的根目录的 inode，且 inode 规范的权限让我们可以读取该 block 的内容(有 r 与 x) ；

2./ 的 block：  
经过上个步骤取得 block 的号码，并找到该内容有 etc/ 目录的 inode 号码 (1912545)；

3.etc/ 的 inode：  
读取 1912545 号 inode 得知 vbird 具有 r 与 x 的权限，因此可以读取 etc/ 的 block 内容；

4.etc/ 的 block：  
经过上个步骤取得 block 号码，并找到该内容有 passwd 文件的 inode 号码 (1914888)；

5.passwd 的 inode：  
读取 1914888 号 inode 得知 vbird 具有 r 的权限，因此可以读取 passwd 的 block 内容；

6.passwd 的 block：  
最后将该 block 内容的数据读出来。

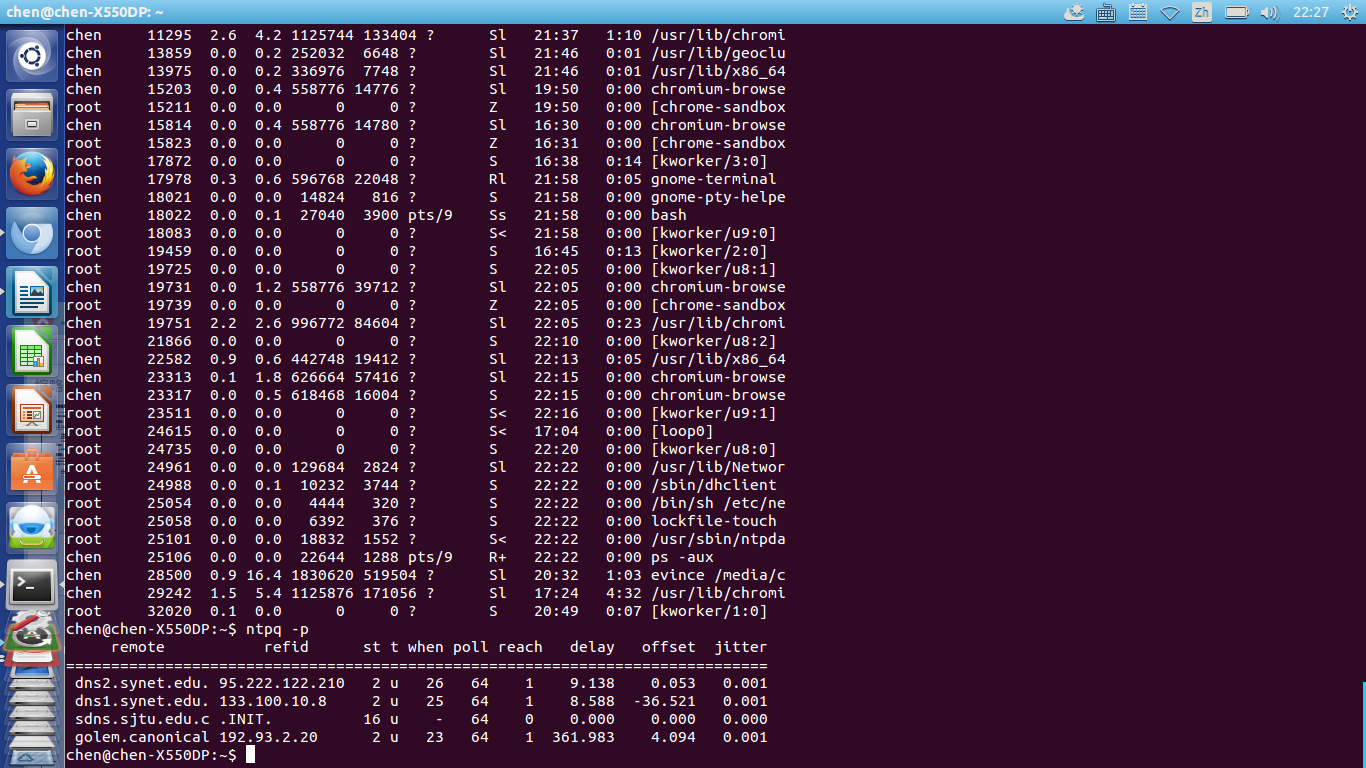
答案： (具体内容如上)

1.由挂载点信息找到根目录下inode号码为2的inode -> 2.相应的block下有etc/目录的inode -> 3.取得其对应inode的block数据 -> 4.找到block内容为passwd的inode -> 5.取得inode下真正的block内容

## 第四题:

**1. 如果以前目录不为空，但是你却将一个文件挂载到其上，会不会影响这个目录？**

答案： 会影响。我们可以将文件挂载在这个目录上面，但是这个目录里面以前的内容会因此而打不开。



## 第六题:

## 第七题. 第八题:

解释:纠结了很久,还是不能写出来多少,查找了网上的答案,但是自己觉得没有多大的意义,我决定先将这部分的原理看一遍总结一下,今天晚上之前要想将所有的程序编写完成那是不太可能完成的,但是我会尽可能地理解.恩,就这样.

### gcc 常用命令：

编译过程是分为四个阶段进行的，即预处理(也称预编译，Preprocessing)、编译(Compilation)、汇编 (Assembly)和连接(Linking)

1. 预编译：预编译又称为预处理，是做些代码文本的替换工作。处理#开头的指令，比如拷贝#include包含的文件代码，#define[宏定义](http://baike.baidu.com/view/2076445.htm)的替换,[条件编译](http://baike.baidu.com/view/1995627.htm)等，就是为[编译](http://baike.baidu.com/view/69568.htm)做的预备工作的阶段，主要处理#开始的预编译指令

2. 编译：(compilation , compile)

1、利用[编译程序](http://baike.baidu.com/view/454895.htm)从源语言编写的[源程序](http://baike.baidu.com/view/546605.htm)产生[目标程序](http://baike.baidu.com/view/1182933.htm)的过程。

2、用[编译程序](http://baike.baidu.com/view/454895.htm)产生[目标程序](http://baike.baidu.com/view/1182933.htm)的动作。 编译就是把高级语言变成计算机可以识别的2进制语言，计算机只认识1和0，[编译程序](http://baike.baidu.com/view/454895.htm)把人们熟悉的语言换成2进制的。

[编译程序](http://baike.baidu.com/view/454895.htm)把一个[源程序](http://baike.baidu.com/view/546605.htm)翻译成[目标程序](http://baike.baidu.com/view/1182933.htm)的工作过程分为五个阶段：[词法分析](http://baike.baidu.com/view/487040.htm)；[语法分析](http://baike.baidu.com/view/487037.htm)；语义检查和[中间代码生成](http://baike.baidu.com/view/487032.htm)；[代码优化](http://baike.baidu.com/view/2136208.htm)；[目标代码](http://baike.baidu.com/view/1272000.htm)生成。主要是进行[词法分析](http://baike.baidu.com/view/487040.htm)和[语法分析](http://baike.baidu.com/view/487037.htm)，又称为[源程序](http://baike.baidu.com/view/546605.htm)分析，分析过程中发现有语法错误，给出提示信息。

Linux下的库文件分为两大类分别是动态链接库（通常以.so结尾）和静态链接库（通常以.a结尾），二者的区别仅在于程序执行时所需的代码是在运行时动态加载的，还是在编译时静态加载的

静态链接与动态链接：

静态链接方法：#pragma comment(lib, "test.lib") ，静态链接的时候，载入代码就会把程序会用到的动态代码或动态代码的地址确定下来  
静态库的链接可以使用静态链接，动态链接库也可以使用这种方法链接导入库

动态链接方法：LoadLibrary()/GetProcessAddress()和FreeLibrary()，使用这种方式的程序并不在一开始就完成动态链接，而是直到真正调用动态库代码时，载入程序才计算(被调用的那部分)动态代码的逻辑地址，然后等到某个时候，程序又需要调用另外某块动态代码时，载入程序又去计算这部分代码的逻辑地址，所以，这种方式使程序初始化时间较短，但运行期间的性能比不上静态链接的程序。

静态库链接时搜索路径顺序：

1. ld会去找GCC命令中的参数-L  
2. 再找gcc的环境变量LIBRARY\_PATH  
3. 再找内定目录 /lib /usr/lib /usr/local/lib 这是当初compile gcc时写在程序内的

动态链接时、执行时搜索路径顺序:

1. 编译目标代码时指定的动态库搜索路径  
2. 环境变量LD\_LIBRARY\_PATH指定的动态库搜索路径  
3. 配置文件/etc/ld.so.conf中指定的动态库搜索路径  
4. 默认的动态库搜索路径/lib  
5. 默认的动态库搜索路径/usr/lib

命令：

1. 一次完成： gcc test.c -o test

2. 预处理： gcc -E test.c -o test.i 或 gcc -E test.c

3.编译为汇编代码 ： gcc -S test.i -o test.s

4.汇编: gcc-c test.s -o test.o

5.连接： gcc test.o -o test

6. 多个程序文件的编译gcc test1.c test2.c -o test

7. 检错 gcc -pedantic illcode.c -o illcode

gcc -Wall illcode.c -o illcode

gcc -Werror test.c -o test（在产生警告的地方停止编译）

8.库文件连接 gcc –c –I /usr/dev/mysql/include test.c –o test.o (编译:include 一般存放头文件)

gcc –L /usr/dev/mysql/lib –lmysqlclient test.o –o test（链接：存放库文件）

9.强制静态链接 gcc –L /usr/dev/mysql/lib –static –lmysqlclient test.o –o test

### 困难：

后面几篇的内容，内容涉及很陌生，现在看不明白，不过通过这两天的准备，我真心学到了一些东西，程序那里也是真心有困难，做了一点简单的，当做是理解的过程，后面的这段时间会再琢磨一下，争取弄明白。