收到的问题：

Q1：代码段中这一句中的0777字段的含义？

A1： 是rwxrwxrwx权限，并且没有特殊权限；更具体地来说，表示文件所有者、文件所有者所在的组的\*用户、\*所有用户，都有权限进行读、写、执行的操作。

Q2：目录文件和文件的权限有区别吗？

A2：如果要查看文件的详细信息，就必须根据inode号码，访问inode节点读取信息。目录文件的读权限（r）和写权限（w），都是针对目录文件本身。由于目录文件内只有文件名和inode号码，所以如果只有读权限，只能获取文件名，无法获取其他信息，因为其他信息都储存在inode节点中，而读取inode节点内的信息需要目录文件的执行权限（x）。

Q3：inode和文件名之间的链接是硬链接还是软连接？

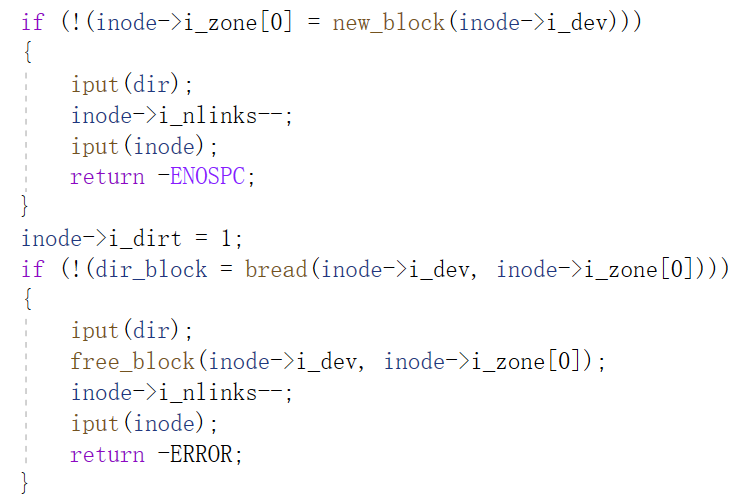
A3：硬链接。当然这个硬链接的数目表示在inode节点中的“链接数”。但是在Linux中也可以创建软链接（指令ln -s），但是不会显示在inode 的“链接数”中，且软链接中，如果文件A软链接了B，那么文件A指向的是B的文件名而不是inode号码。

Q4：在mkdir代码段中详细讲一下bread()函数

A4：参数包括块设备号和数据块号，先调用getblk来获得buffer\_head，bh是有效的，直接返回，bh无效，则用ll\_rw\_block调用底层函数来读取数据，等待读取完成后bh有效返回。若bh无效，则释放bh，返回null。

Q5：文件系统的系统调用可以简单讲一下吗？

A5：在Linux 系统中，read()和write()分别是读和写文件的系统调用，他们既可以被用来读写普通文件，也可以读写目录，块设备文件，字符设备文件和管道文件，可谓全能。事实上，read()和write()本身并不直接负责多种不同类型的文件的读写操作，它们仅仅是根据文件句柄在filp 中找到对应项，获取文件的inode，判定它的类型。最后根据类型来调用相应的函数即可，这就是这两个系统调用的全部逻辑。

Q6：在mkdir的时候能详细讲一下下面的含义吗？

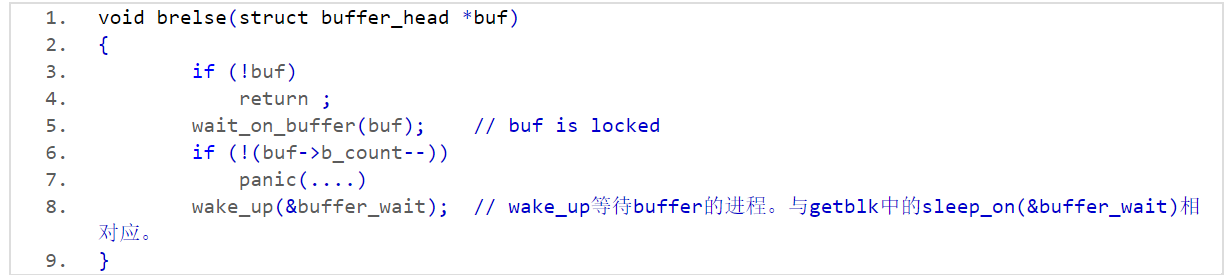
A6：从设备上读取新申请的磁盘块（目的是把对应块放到高速缓冲区中），如果出错，则放回对应目录的inode；释放申请的磁盘块；复位inode连接计数；放回inode；返回出错码退出。

Q7：inode号是可重复的吗？

A7：对于一个文件来说，inode号是唯一的，但是inode是可以复用的。由于每个文件都必须有一个inode，因此有可能发生inode已经用光，但是硬盘还未存满的情况。这时，就无法在硬盘上创建新文件。当文件被删除的时候，该文件的inode将会被新文件使用。对于轮询文件，旧文件被删除之后，新文件立马创建了，此时两个文件的inode可能是一样的。

Q8：详细解释一下brelse（）函数

A8：释放buffer\_head的函数。



Q9：mkdir运行的时候是直接访问磁盘上的数据吗？

A9：不是的，文件的读取和写入这类操作都是通过高速缓冲区来实现对接的，而不是系统直接对磁盘进行操作。

Q10：在你讲的文件系统中，Linux0.11为什么只支持8个超级块？这个8是由什么规定的？

A10：Linux0.11只支持8个超级快同时挂载，而不是说只支持8个超级块，这是因为Linux0.11在内存中为超级块开辟的空间只能容纳8个超级块，这个8的定义在include/linux/fs.h中。

Q11：在Linux0.11中一个文件最大的大小是多少？

A11：最多占用7+512+512×512 个逻辑块，这是由inode中的i\_zone[]所决定的。（i\_zone[9]中直接(0-6)、间接(7)或双重间接(8)逻辑块号）

Q12：在代码段中，namei()函数的具体内容是什么？能讲解一下吗

A12：namei() 相当于基于get\_dir()继续完成通过路径最终文件和目录的inode 指针的任务。首先找到最终文件的所属目录的inode 指针，如果路径的最后一个字符是/，直接返回inode 指针，否则使用find\_entry()函数找到最终文件，获取对应的inode 指针，修改inode 的最近访问时间，然后返回inode 指针。

Q13：具体解释一下find\_entry()函数

A13：find\_entry() 在目录中按文件名搜索，返回指向含有目录项的缓冲块的指针和指向目录项的指针。首先根据目录的 inode 记录的文件大小和一个目录项的大小（16 字节）计算出目录中的项数。然后处理当前进程可能被设置的 chroot 机制。如果当前工作目录是一个文件系统的根目录，而该函数搜索的是..的话，将put 并且重新设置需要查找的目录的inode，并使它的引用计数加 1（如果当前工作目录并非一个文件系统的根目录并且搜索的是..的话，将没有必要特殊处理）。最后，逐项搜索目录的目录项，并且只在一个目录的逻辑块被查找完毕后才载入下一个逻辑块，而不是一下子全部载入。如果找到匹配项，立即返回。

Q14：解释一下getblk（）函数

A14：getblk函数首先调用get\_hash\_table函数，从hash表中查找指定的（dev，block）是否已经在缓冲区中。

若没有在缓冲区中，则会在free\_list中寻找一个空的buffer\_head。

空的buffer\_head的意思是指buffer\_head.b\_count值为0，没有被任何进程引用。

但该buffer\_head指向的缓冲块可能是b\_dirt=1和b\_lock=1的。根据这两个值，对每个buffer\_head定义了一个优先级。

#define BADNESS(bh) (((bh)->b\_dirt 这个宏表明：但b\_dirt和b\_lock都为1时，该buffer\_head为非常BADNESS的。优先选择b\_dirt=0和b\_lock=0的buffer\_head.

当找到一个buffer\_head之后，如果它是lock的，就需要等待解锁，如果它是dirt的，则需要将缓冲区的内容写入磁盘。

若没有找到适合的buffer\_head，则调用sleep\_on(&buffer\_wait)进入睡眠。

若最终得到一个buffer\_head后：

bh->b\_count = 1; // 占用该缓冲块

bh->b\_dirt = 0; // 干净的

bh->b\_uptodate = 0; // 复位

然后将buffer\_head从hash表取出，并移到free\_list链表的最后去（最近最少使用算法）

Q15：inode节点的大小是固定的吗？

A15：每个inode节点的大小，一般是128字节或256字节。inode节点的总数，在格式化时就给定，一般是每1KB或每2KB就设置一个inode。假定在一块1GB的硬盘中，每个inode节点的大小为128字节，每1KB就设置一个inode，那么inode table的大小就会达到128MB，占整块硬盘的12.8%。当然这个看系统版本而定，如果查看每个硬盘分区的inode总数以及已经使用的数量，可以使用df指令来查看。如果是想查看每个inode节点的大小，可以用指令“sudo dumpe2fs -h /dev/hda | grep "Inode size"”

Q16：在mkdir代码段中的if(!(dir\_block = bread(inode->i\_dev,inode->i\_zone[0])))开始的代码段的具体意思没怎么懂。

A16：这个问题的答案同Q6