File System 实验报告

专　业：软　件　工　程

指　导　老　师：王　冬　青

年　级：大　学　二　年　级

学　号：2153061

姓　名：谢　嘉　麒

1. Large files (moderate)

1.1.实验目的

本实验要求增加xv6文件的最大大小。目前xv6文件被限制为268个块，或268*BSIZE字节（xv6中BSIZE为1024）。这一限制是因为xv6的inode包含12个"直接"块号和一个"一级间接"块号，后者指向一个块，该块最多可以容纳256个更多的块号，总共有12+256=268个块。

1.2.实验步骤

1.2.1.修改fs.h

根据xv6实验手册推荐的方案，使用三级索引，共有11个直接索引，1个间接索引块和1个二级间接索引块，故总共支持文件大小为11+256+256*256=65803块。

因此可以修改fs.h，使其符合三级索引：

```
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NTWOINDIRECT ((NINDIRECT)*(NINDIRECT))
#define MAXFILE (NDIRECT + NINDIRECT + NTWOINDIRECT)

struct dinode {
  short type;
  short major;
  short minor;
  short nlink;
  uint size;
  uint addrs[NDIRECT+2];
};
```

1.2.2.修改bmap

修改kernel/fs.c文件中的bmap()函数，具体方法如下：

（1）这一函数从inode映射到磁盘块，设定addr和*a作为存储块的地址；

（2）设定struct buf类型的指针bp作为指向缓冲区的指针，用于读取磁盘块；

（3）当bn<NDIRECT，即bn在直接块的范围内，则获取直接块的地址，并在地址为0时分配一个块，返回直接块的地址；

（4）利用bn -= NDIRECT用于计算一级间接块中的偏移量，将一级间接块的数据指针bp-data赋予a；

（5）若addr = a[bn] = 0，则获取数据块对应的地址，并通过log_write(bp)写回磁盘，之后释放一级间接块（使用brelse(bp)）并返回数据块地址；

（6）利用bn -= NINDIRECT计算二级间接块中的偏移量；

（7）当bn < NTWOINDIRECT，加载二级间接块，并在地址为0时分配一个块；

（8）使用bread读取二级间接块，赋予bp_2；

（9）利用bp_2->data即为二级间接块的数据指针，进而判断addr=a_2[bn/NINDIRECT]=0，获取一级间接块地址，并在地址为0时分配块；

（10）更新二级间接块中的地址，使用log_write(bp_2)写回磁盘，释放二级间接块（brelse(bp_2)）；

（11）继续读取一级间接块并获取数据指针，同上方法依次执行获取数据块地址，分配块，更新一级间接块中的地址，写回磁盘，释放一级间接块并返回数据块地址；

（12）以上条件均不满足则发出panic；

1.2.3.修改itrunc()

这一函数用于截断与inode ip相关联的文件并且释放分配给文件的所有数据块；

（1）分别设置struct buf，* bp，* bp_2来指向缓冲区，读取磁盘块；

（2）设置uint * a，* a_2来指向数据；

（3）之后思路相似，先遍历并释放所有直接块；

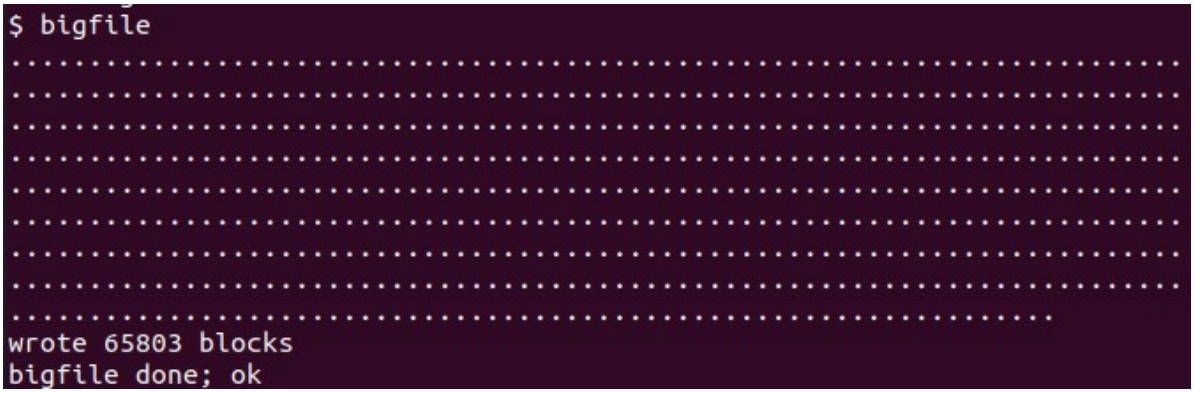
（4）检查是否有一级间接块，有则释放其中数据块并释放一级间接块；

（5）检查是否有二级间接块，有则遍历一级间接块并释放数据块，然后释放一级间接块，再释放二级间接块；

（6）最后将inode大小设置为0并更新。

1.2.4.编译与检测

完成以上步骤后，在xv6终端下执行make qemu进行编译，输入bigfile进行检测，得到预期结果：



1.3.实验中遇到的问题和解决方法

1.3.1.理解间接块的设计与使用

本实验要求更改xv6文件系统来支持二级间接块，因此需要对这一概念有一定的了解。实际上这一内容的学习与实现可以参考多级页表，两者异曲同工。同时要明确代码实现需要多次迭代，知晓直接块和一级间接块该如何查找数据块即可。

1.4.实验心得

通过本次实验，笔者在准备阶段了解了间接块和大文件系统的具体实现，同时也在学习过程中借鉴多级页表的理解过程，对知识的巩固和体系形成有很大裨益。

2. Symbolic links (moderate)

2.1.实验目的

本实验中，需要为xv6添加符号链接。符号链接（或软链接）通过路径名引用一个链接文件；当打开符号链接时，内核会跟随链接到所引用的文件。符号链接类似于硬链接，但硬链接只能指向同一磁盘上的文件，而符号链接可以跨越磁盘设备。

实验要求实现symlink(char *target, char *path)系统调用，在路径path创建一个新的符号链接，指向由target命名的文件。

2.2.实验步骤

2.2.1.实现symlink系统调用

本系统调用用于创建一个符号链接文件，仿照已有的文件系统调用代码即可，此外需要处理日志。

（1）系统调用首先从用户空间获取两个参数target和path，分别制定链接的目标文件名和符号链接文件的路径名；

（2）调用begin_op()开始文件系统操作，来确保操作文件系统期间不会发生其他文件系统操作；

（3）在给定的path路径上创建一个新的符号链接文件，并返回该文件对应的inode指针，调用create()函数，创建一个类型为T_SYMLINK的inode，用于表示符号链接文件；

（4）创建文件失败（返回的inode指针为0），则释放之前开始的文件系统操作，返回-1表示创建符号链接失败；

（5）否则在符号链接文件的数据块中存储target内容，即目标文件名。调用writei()函数将target内容写入符号链接文件的数据块；

（6）写入字节数若少于MAXPATH，说明写入不成功，返回-1表示创建符号链接失败；

（7）若写入成功，则释放并解锁符号链接文件的inode，结束文件系统操作，并返回0表示符号链接创建成功。

2.2.2.修改open系统调用

需要在open中添加功能，用于在打开文件时对符号链接进行解析（follow）操作，即符号链接文件的内容获取实际目标文件的inode，并将其返回。

（1）检查inode是否为符号链接文件T_SYMLINK，同时打开文件模式不包含0_NOFOLLOW，则要对符号链接进行解析；

（2）定义depth规定解析符号链接的最大深度，创建数组target存储读取到的符号链接的目标路径；

（3）循环解析，每次循环中从符号链接文件的数据块中读取路径target，调用readi()读取数据块；

（4）调用iunlockput(ip)释放并解锁符号链接文件，根据target调用namei()查找目标文件inode，找到则返回给ip，继续循环解析；

（5）未找到则释放文件系统操作，返回-1表示解析失败；

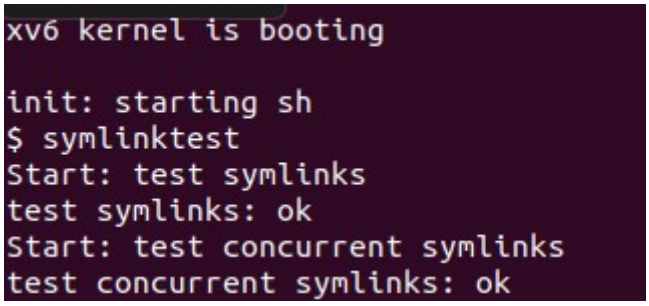
（6）找到目标文件后，用ilock(ip)锁定inode，防止并发操作；

（7）检查目标文件类型是否为符号链接，是则还需要继续解析目标文件的目标路径，继续循环；

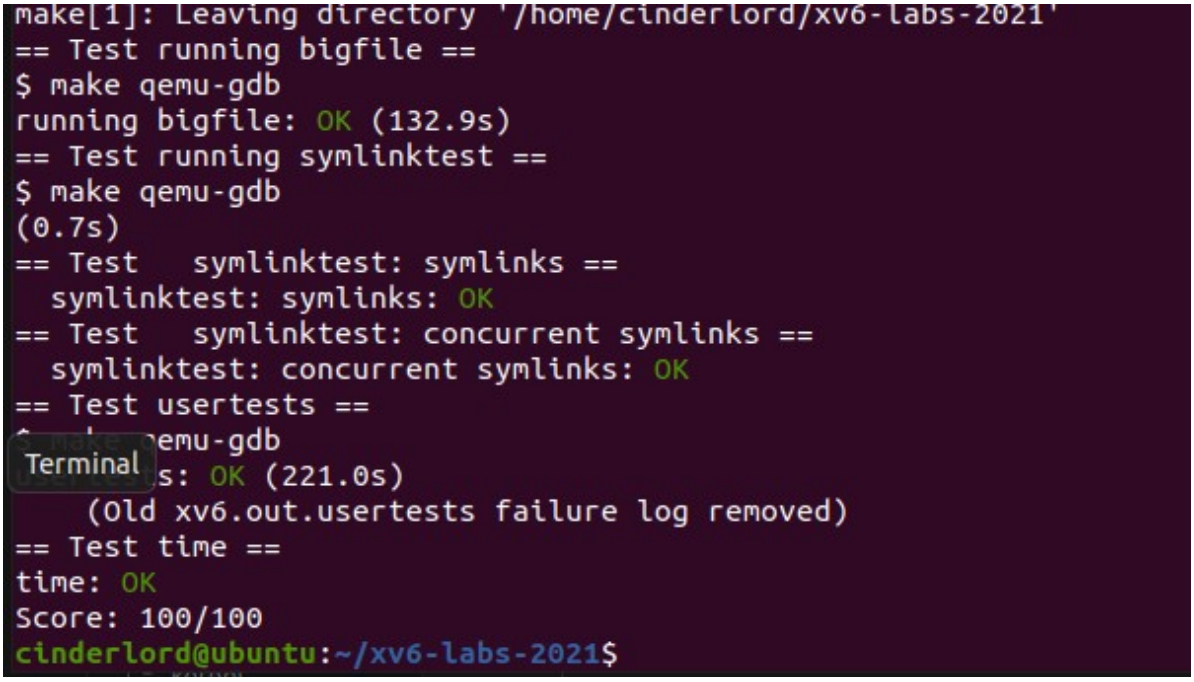
（8）不是符号链接，说明已经得到最终目标文件，可以结束循环解析操作。或循环达到最大深度，也结束循环操作；

2.2.3.编译与检测

完成以上步骤后，在xv6系统中编译并执行symlinktest进行检测，得到预期结果如下：



执行make grade对整个实验进行评分：



2.3.实验中遇到的问题和解决方法

2.3.1.创建符号链接文件并循环解决问题

实验中主要需要学习如何创建符号链接文件，如何写入target路径，何时该进行文件解析，该怎样读出target路径，该如何解析target路径以及解析的终止条件。这几个问题通过阅读文件系统相关源代码和xv6实验手册即可解决；

同时要注意控制解析的最大深度，防止循环出错

2.4.实验心得

通过本实验，笔者理解乐路径名查找的工作原理，同时也对符号链接这一功能有了一定的认识。再具体的代码实现过程中，包括文件系统操作加锁，防止文件并发操作等细节操作，都警示笔者在进行系统调用实现时需要考虑这些基础内容。