103011227 周延儒

[20 points] 11.2 Contrast the performance of the **three** techniques for allocating disk blocks (contiguous, linked, and indexed) for both **sequential and random** file access. You must elaborate to receive full credit.

1. Contiguous Sequential - Works very well if the file is **stored contiguously. And as for Sequential access - Simply involves traversing the contiguous disk blocks. •** Contiguous Random -Works very well as you can easily determine **the adjacent disk block** containing the position you wish to seek to.
2. Linked Sequential - **Satisfactory** as you are simply following the **links from one block to the next.**
   Linked Random - **Poor** as it may require traverse the links to several disk blocks until you arrive at the intended seek point of the file.
3. Indexed Sequential - **Works well** as sequential access simply involves **sequentially accessing each index.**
   Indexed Random - **Works well** as it is easy to **determine the index** associated with the disk block containing the position you wish to seek to

[10 points] 11.8 Consider a file system that uses inodes to represent files. Disk blocks are 8 KB in size, and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, as well as single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?

(12 * 8 /KB/) + (2048 * 8 /KB) + (2048 * 2048 * 8 /KB/) +

(2048 * 2048 * 2048 * 8 /KB) = **64 terabytes**

2.1 [5 points]

What are the inode values of `file1.txt` and `file2.txt`? Are they the same or different? Do the two files have the same—or different— contents?

```
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ln file1.txt file2.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls
file1.txt       file2.txt       file3.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls -li
total 24
8624860860 -rw-r--r--  2 Ruchou  staff  16 Nov 29 11:03 file1.txt
8624860860 -rw-r--r--  2 Ruchou  staff  16 Nov 29 11:03 file2.txt
8624860888 -rw-r--r--  1 Ruchou  staff  15 Nov 29 11:04 file3.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$
```

yes file1.txt and file2.txt share the **same** inode value of 8624860860.

And they share the same contents

2.2 [5 points]

Next, edit `file2.txt` and change its contents. After you have done so, examine the contents of `file1.txt`. Are the contents of `file1.txt` and `file2.txt` the same or different?

```
this is a book                                                  this is a book
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ echo 'hello world'> file2.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ diff file1.txt -y file2.txt
hello world                                                     hello world
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$
```

Well, they contents will be **the same**

This is because they share the same data on the disk. Changes to the data of either one of these files will change the contents of the other.

.

2.3 [5 points]

Next, enter the following command which removes `file1.txt`:

```
rm file1.txt
```

Does `file2.txt` still exist as well?

```
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ rm file1.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls
file2.txt       file3.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$
```

Then, the file2.txt still exist.

This is because if we delete one of the files, we're deleting one of the **links** to the data. Because we created another link manually, we still have a pointer to that data

2.4 [5 points]

Now examine the man pages for both the `rm` and `unlink` commands. Afterwards, remove `file2.txt` by entering the command

```
strace rm file2.txt
```

The `strace` command traces the execution of system calls as the command `rm file2.txt` is run. What system call is used for removing `file2.txt`?

In fact, while **rm uses the unlink() system call for its main purpose**, it uses many more calls as well.

```
("/bin/rm", ["rm", "file2.txt"], [/* 71 vars */]) = 0
brk(NULL)                                       = 0x1a27000
access("/etc/ld.so.nohwcap", F_OK)          = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f23920ad000
access("/etc/ld.so.preload", R_OK)          = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=107829, ...}) = 0
mmap(NULL, 107829, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2392092000
close(3)                                        = 0
access("/etc/ld.so.nohwcap", F_OK)          = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1868984, ...}) = 0
mmap(NULL, 3971488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE,
3, 0) = 0x7f2391ac0000
mprotect(0x7f2391c80000, 2097152, PROT_NONE) = 0
mmap(0x7f2391e80000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c0000) = 0x7f2391e80000
mmap(0x7f2391e86000, 14752, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2391e86000
close(3)                                        = 0
```

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2392091000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2392090000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f239208f000
arch_prctl(ARCH_SET_FS, 0x7f2392090700) = 0
mprotect(0x7f2391e80000, 16384, PROT_READ) = 0
mprotect(0x60d000, 4096, PROT_READ)        = 0
mprotect(0x7f23920af000, 4096, PROT_READ) = 0
munmap(0x7f2392092000, 107829)             = 0
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=10505632, ...}) = 0
mmap(NULL, 10505632, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f23910bb000
close(3)                                   = 0
brk(NULL)                                   = 0x1a27000
brk(0x1a48000)                              = 0x1a48000
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0664, st_size=4, ...}, AT_SYMLINK_NOFOLLOW) = 0
geteuid()                                  = 1000
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0664, st_size=4, ...}, AT_SYMLINK_NOFOLLOW) = 0
faccessat(AT_FDCWD, "file2.txt", W_OK)    = 0
unlinkat(AT_FDCWD, "file2.txt", 0)         = 0
lseek(0, 0, SEEK_CUR)                       = -1 ESPIPE (Illegal seek)
close(0)                                   = 0
close(1)                                   = 0
close(2)                                   = 0
exit_group(0)                              = ?
+++ exited with 0 ++
```

2.5 [10 points]

A soft link (or symbolic link) creates a new file that “points” to the name of the file it is linking to. In the source code available with this text, create a soft link to `file3.txt` by entering the

following command:

```
ln -s file3.txt file4.txt
```

After you have done so, obtain the inode numbers of `file3.txt` and `file4.txt` using the command

```
ls -li file*.txt
```

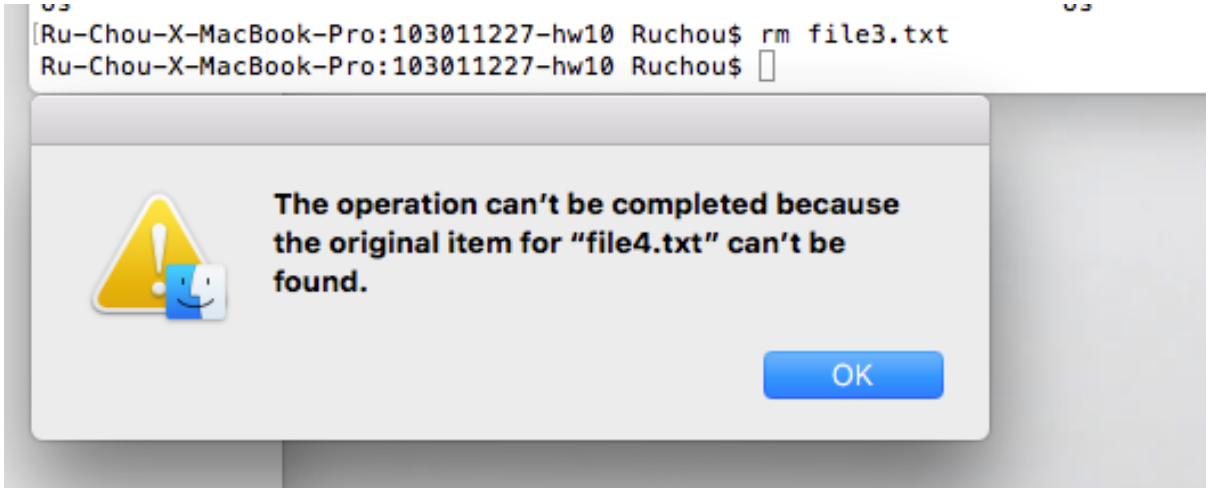
Are the inodes the same, or is each unique? Next, edit the contents of `file4.txt`. Have the contents of `file3.txt` been altered as well? Last, delete `file3.txt`. After you have done so, explain what happens when you attempt to edit `file4.txt`.

```
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls
file3.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ln -s file3.txt file4.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls
file3.txt       file4.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ ls -li
total 8
8624864453 -rw-r--r--  1 Ruchou  staff  13 Nov 29 11:34 file3.txt
8624864509 lrwxr-xr-x  1 Ruchou  staff   9 Nov 29 11:35 file4.txt -> file3.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$
```

First of all, the inodes are **different**.

```
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ echo 'os' > file4.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ diff file3.txt file4.txt
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$ diff file3.txt -y file4.txt
os                                                              os
Ru-Chou-X-MacBook-Pro:103011227-hw10 Ruchou$
```

Secondly, after modifying the content of file4.txt, and the corresponding file3.txt is **changed too.**



Lastly, after deleting the file3.txt, the file4.txt can not be edited. This is because a symbolic

link is like a shortcut from one file to another. The contents of a symbolic link are the address of the actual file or folder that is being linked to.

After entering the rm – removes each given FILE including symbolic links , unlike hard links, removing the file (or directory) that a symlink points to will break the link.
It's a broken symlink, however — a symbolic link which points to something that no longer exists.

Another difference between a hard link and a symbolic link is that a hard link must be created against a file that already exists whereas a soft link can be created in advance of the file it is pointing to existing