系統環境：VMware, Linux, Ubuntu 18.04

- Problem 11.13

(1) file1.txt 和 file2.txt 的 inode 是一樣的，並且有相同的內容。

```
yuwei@ubuntu:~/Desktop/folder$ gedit file1.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li file1.txt
1314727 -rw-rw-r-- 1 yuwei yuwei 6 Jun  1 07:03 file1.txt
yuwei@ubuntu:~/Desktop/folder$ ln file1.txt file2.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li *.txt
1314727 -rw-rw-r-- 2 yuwei yuwei 6 Jun  1 07:03 file1.txt
1314727 -rw-rw-r-- 2 yuwei yuwei 6 Jun  1 07:03 file2.txt
yuwei@ubuntu:~/Desktop/folder$ cat file1.txt
Apple
yuwei@ubuntu:~/Desktop/folder$ cat file2.txt
Apple
```

(2) 修改 file2.txt 也會讓 file1.txt 的內容更新，刪除 file1.txt 後 file2.txt 仍能正常使用

```
yuwei@ubuntu:~/Desktop/folder$ gedit file1.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li file1.txt
1314727 -rw-rw-r-- 1 yuwei yuwei 6 Jun  1 07:03 file1.txt
yuwei@ubuntu:~/Desktop/folder$ ln file1.txt file2.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li *.txt
1314727 -rw-rw-r-- 2 yuwei yuwei 6 Jun  1 07:03 file1.txt
1314727 -rw-rw-r-- 2 yuwei yuwei 6 Jun  1 07:03 file2.txt
yuwei@ubuntu:~/Desktop/folder$ cat file1.txt
Apple
yuwei@ubuntu:~/Desktop/folder$ cat file2.txt
Apple
yuwei@ubuntu:~/Desktop/folder$ gedit file2.txt
yuwei@ubuntu:~/Desktop/folder$ cat file2.txt
Banana
yuwei@ubuntu:~/Desktop/folder$ cat file1.txt
Banana
yuwei@ubuntu:~/Desktop/folder$ rm file1.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li *.txt
1314727 -rw-rw-r-- 1 yuwei yuwei 7 Jun  1 07:07 file2.txt
```

使用 strace 查看移除 file2.txt 所呼叫到的 system call，其中有用於檢查和訪問文件、動態鏈接庫相關操作，使得執行 rm 命令能夠正確地找到並刪除文件。

```
yuwei@ubuntu:~/Desktop/folder$ strace rm file2.txt
execve("/bin/rm", ["rm", "file2.txt"], 0x7ffdf8944b58 /* 53 vars */) = 0
brk(NULL)                               = 0x55a230b3e000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=78175, ...}) = 0
mmap(NULL, 78175, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1b7b6a3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/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\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1b7b6a1000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1b7b09d000
mprotect(0x7f1b7b284000, 2097152, PROT_NONE) = 0
mmap(0x7f1b7b484000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f1b7b484000
mmap(0x7f1b7b48a000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1b7b48a000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f1b7b6a2540) = 0
mprotect(0x7f1b7b484000, 16384, PROT_READ) = 0
mprotect(0x55a22fe53000, 4096, PROT_READ) = 0
mprotect(0x7f1b7b6b7000, 4096, PROT_READ) = 0
munmap(0x7f1b7b6a3000, 78175)           = 0
brk(NULL)                               = 0x55a230b3e000
brk(0x55a230b5f000)                     = 0x55a230b5f000
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=3004224, ...}) = 0
mmap(NULL, 3004224, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1b7adbf000
close(3)                                = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...}) = 0
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0664, st_size=7, ...}, AT_SYMLINK_NOFOLLOW) = 0
geteuid()                               = 1000
newfstatat(AT_FDCWD, "file2.txt", {st_mode=S_IFREG|0664, st_size=7, ...}, 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 +++
```

(3) 利用弱連結的檔案結構，連結的兩個檔案 inode 並不會相同，修改 file4.txt 也會讓 file3.txt 的內容一併更新，刪除 file3.txt 會讓 file4.txt 的內容遺失，由於弱連結指向的檔案被刪除，因此它變為唯獨且內容遺失。

```
yuwei@ubuntu:~/Desktop/folder$ gedit file3.txt
yuwei@ubuntu:~/Desktop/folder$ cat file3.txt
Cherry
yuwei@ubuntu:~/Desktop/folder$ ln -s file3.txt file4.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li *.txt
1314727 -rw-rw-r-- 1 yuwei yuwei 7 Jun  1 07:55 file3.txt
1315158 lrwxrwxrwx 1 yuwei yuwei 9 Jun  1 07:55 file4.txt -> file3.txt
yuwei@ubuntu:~/Desktop/folder$ gedit file4.txt
yuwei@ubuntu:~/Desktop/folder$ cat file4.txt
Dragonfruit
yuwei@ubuntu:~/Desktop/folder$ cat file3.txt
Dragonfruit
yuwei@ubuntu:~/Desktop/folder$ rm file3.txt
yuwei@ubuntu:~/Desktop/folder$ ls -li *.txt
1315158 lrwxrwxrwx 1 yuwei yuwei 9 Jun  1 07:55 file4.txt -> file3.txt
yuwei@ubuntu:~/Desktop/folder$ cat file4.txt
cat: file4.txt: No such file or directory
```

- Problem 12.16

編寫一個能根據給予的請求序列，在不同的磁碟排程演算法（FCFS、SSTF、SCAN、C-SCAN）中，磁碟頭需要移動的距離（又稱耗時），在實現後能發現，SSTF 和 SCAN 在大多數情況下的效率表現較佳。

FCFS：依照請求序列的順序，header 會依序移動至指定的 cylinder，並累加移動距離，得到以 FCFS 演算法處理的 header 移動路徑長度。

SSTF：先標記請求序列的標頭索引 header，將請求序列依照大小排序，並定義 2 個距離 header 最近的索引（forward、backward），取得兩個方向需要的移動距離，header 以計算後最短移動距離的方向移動，累加 header 移動的距離，header 如果往 forward 跑，forward - 1；header 如果往 backward 跑，backward + 1，直到所有的 cylinder 請求都被執行過，得到以 SSTF 演算法處理的 header 移動路徑長度，是四種方法中在絕大部分情況下最有效率的排程演算法。

SCAN：先標記請求序列的標頭索引 header，將請求序列依照大小排序，再將排序後的序列，以 header 作為序列分割的索引，將 header 包含的部分

序列作為新序列，再將剩餘部份的序列翻轉後新增在新序列後面，得到 SCAN 演算法中 header 會依序執行的 SCAN 序列，舉例：

假設 header 在 2150 cylinder：

已排序的序列：356 544 1212 1523 1618 2069 2150 2296 2800 3681 4965

SCAN 序列：2150 2296 2800 3681 4965 2069 1618 1523 1212 544 356

header 再依序以新序列陸續移動，並累加 header 的移動距離，得到以 SCAN 演算法處理的 header 移動路徑長度。

C-SCAN：先標記請求序列的標頭索引 header，將請求序列依照大小排序，再將排序後的序列，以 header 作為序列分割的索引，將 header 包含的部分序列作為新序列，先在新序列後方插入一個零，代表 C-SCAN 從右邊緣一移動至左邊緣返回移動行為，再將剩餘部分的序列直接串接在新序列後面，得到在 C-SCAN 演算法中 header 會依序執行的 C-SCAN 序列，舉例：

假設 header 在 2150 cylinder：

已排序的序列：356 544 1212 1523 1618 2069 2150 2296 2800 3681 4965

C-SCAN 序列：2150 2296 2800 3681 4965 0 356 544 1212 1523 1618 2069

header 再依序以新序列陸續移動，並累加 header 的移動距離，得到以 C-SCAN 演算法處理的 header 移動路徑長度。

程式碼：Source code\Chap12.16\disk_scheduling.c

```
yuwei@ubuntu:~/Desktop$ gcc -o disk_scheduling disk_scheduling.c
yuwei@ubuntu:~/Desktop$ ./disk_scheduling 2150 2069 1212 2296 2800 544 1618 356 1523 4965 3681
FCFS    2150 2069 1212 2296 2800 544 1618 356 1523 4965 3681
FCFS total distance: 13011
SSTF    2150 2069 2296 2800 3681 4965 1618 1523 1212 544 356
SSTF total distance: 7586
SCAN    2150 2296 2800 3681 4965 2069 1618 1523 1212 544 356
SCAN total distance: 7424
C-SCAN  2150 2296 2800 3681 4965 0 356 544 1212 1523 1618 2069
C-SCAN total distance: 9849
```