Assignment

Versioning Copy-On-Write File System (vCowFS)

GROUP 3

รายงาน

วิชา 01076259 Operating Systems

เสนอ

ดร.อักฤทธิ์ สังข์เพ็ชร

ดร.อรทัย สังข์เพ็ชร

จัดทำโดย

1. นาย แทนไท	เอียการนา	56010492
2. นาย พงษ์ศักดิ์	สงวนวงษ์	57010821
3. นาย พรเทพ	แซ่อึ้ง	57010836
4. นาย ภานุวัฒน์	เอมอำไพวงศ์	57010978
5. น.ส. วรัญญา	กิจประไพอำพล	57011116
6. น.ส. วิรชา	เลาหพูนรังษี	57011180
7. นาย ศรัทธาธรรม์	จันทร์ชาตรี	57011220
8. นาย ศวีระ	อภินทนาพงศ์	57011229
9. นาย สิรวิชญ์	วนรัฐิกาล	57011363
10. นาย อนุรักษ์	จันนาวัน	57011470

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

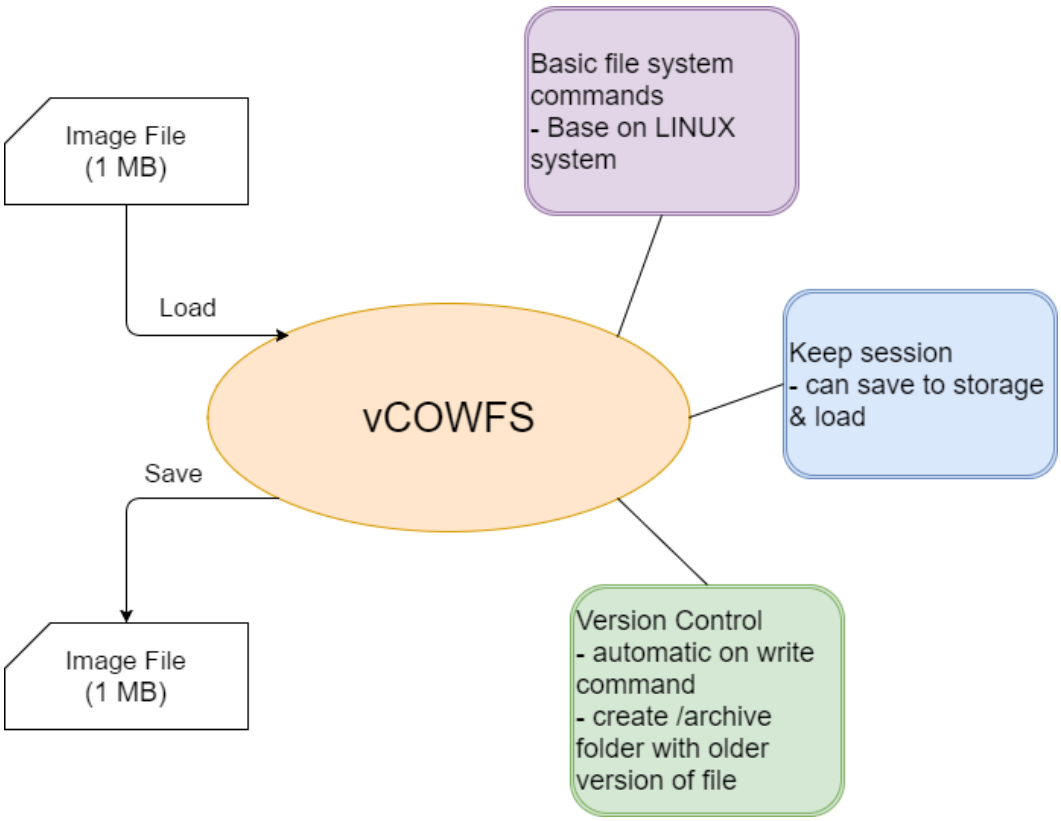
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Preface

ในปัจจุบันนี้คอมพิวเตอร์ได้เข้ามามีบทบาทต่อการดำเนินชีวิตประจำวันของเราเป็นอย่างมาก และหัวใจสำคัญของเครื่องคอมพิวเตอร์ทุกๆเครื่องก็คือโปรแกรมที่เรียกว่า ระบบปฏิบัติการ ซึ่งเป็นโปรแกรมที่มีความซับซ้อนสูงมากและมีการทำงานร่วมกับอุปกรณ์ต่างๆที่ติดตั้งในเครื่อง ซึ่งจากเหตุผลที่ว่าระบบปฏิบัติการนั้นมีความสำคัญและเป็นหัวใจของคอมพิวเตอร์ ทำให้ต้องมีการรองรับปัญหาที่อาจเกิดขึ้นในหลายๆประการ อาทิ เช่น ไฟล์ของระบบปฏิบัติการเสียหายทำให้ไม่สามารถเปิดเครื่องได้, การที่ระบบติดไวรัสแล้วต้องการกู้ระบบคืน ซึ่งรายงานฉบับนี้จะแสดงให้เห็นถึงการทำงานของระบบดังกล่าวซึ่งเป็นการเขียนไฟล์ระบบสำรองไว้เป็นช่วงเวลา ทำให้ผู้ใช้สามารถเรียกใช้ไฟล์เวอร์ชั่นเก่าได้ โดยใช้หลักการทำงานที่เรียกว่า vCowFS ซึ่งจะทำให้ระบบปฏิบัติการสามารถเขียนไฟล์และอ่านไฟล์ได้

คณะผู้จัดทำ

Contents

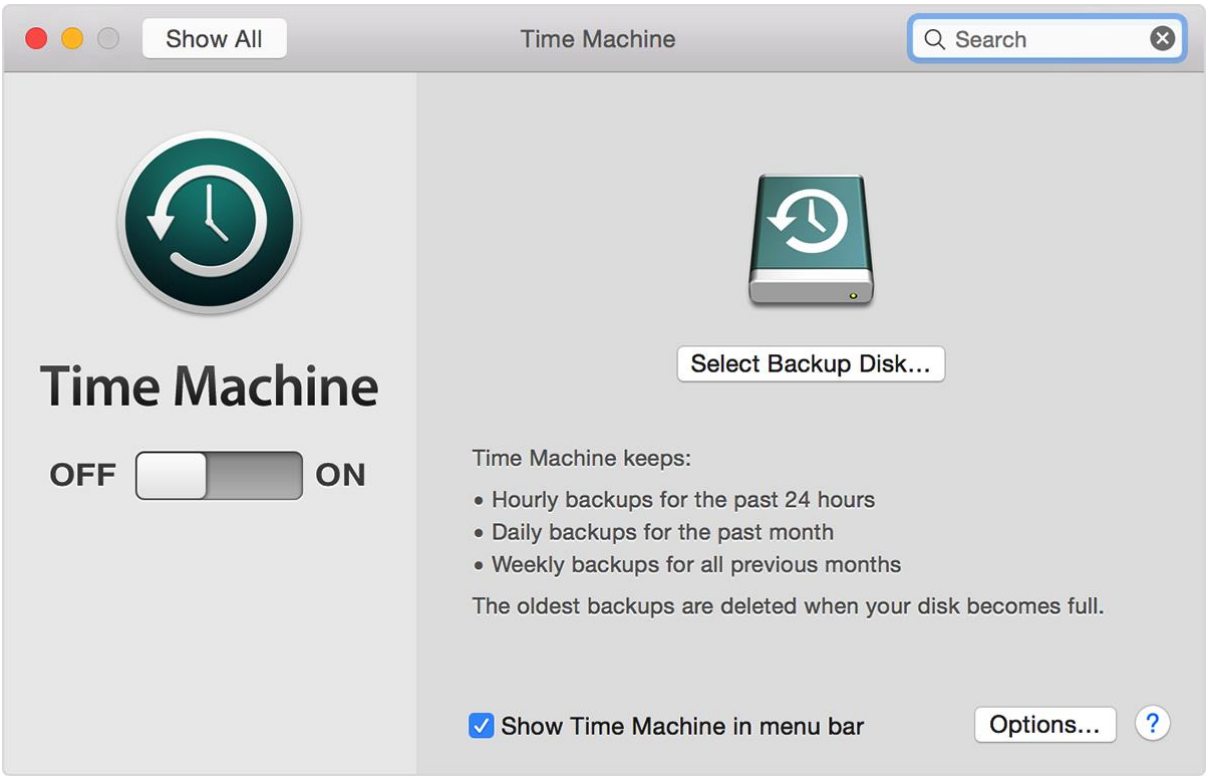


Chapter 1 Design Process

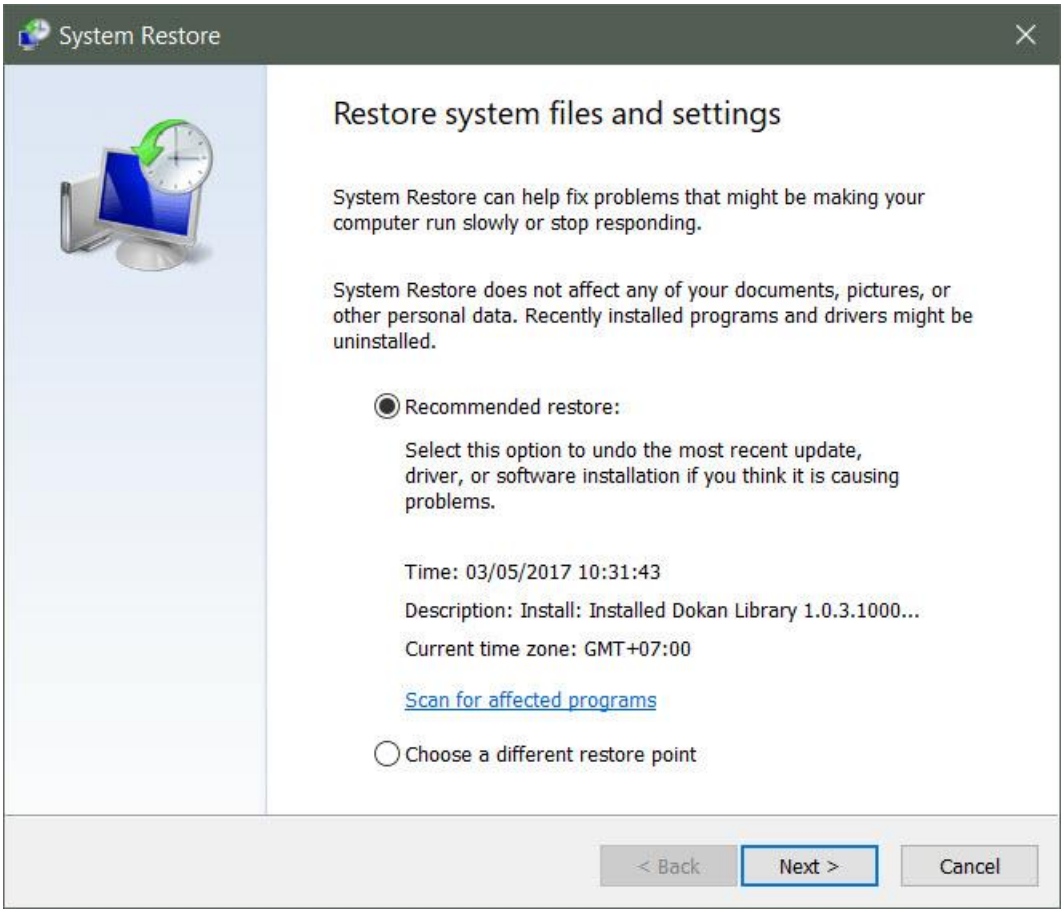
1.1 ปัญหา

ในระบบปฏิบัติการต่างๆ นั้นจะมีสิ่งที่สำคัญที่จะทำให้ระบบสามารถดำเนินการทำงานต่อไปได้ โดยสิ่งนั้นจะเรียกว่า File System โดยไฟล์ดังกล่าวจะทำงานอยู่ใน Kernel ของระบบปฏิบัติการซึ่งผู้ใช้งานทั่วไปไม่สามารถเข้าถึงได้ ซึ่งไฟล์ดังกล่าวมีความสำคัญเป็นอย่างมาก หากไฟล์ดังกล่าวเกิดความเสียหายหรือมีข้อผิดพลาดขึ้นมาจากส่งผลให้ระบบเสียหายและไม่สามารถทำงานต่อไปได้

การทำ vCowFS นั้นสามารถช่วยลดโอกาสเกิดปัญหาดังกล่าวได้โดยให้ไฟล์ระบบทำการเขียนไฟล์เป็น Version เพื่อให้สามารถเรียกใช้งานไฟล์รุ่นก่อนหน้าได้ เช่นในระบบปฏิบัติการ MacOS X นั้นมีระบบที่เรียกว่า Time Machine หรือใน Windows มีระบบที่เรียกว่า System Restore ที่สามารถย้อนไฟล์ดังกล่าวให้กลับมามีสภาพที่สมบูรณ์ได้



ภาพแสดงระบบ Time Machine ของระบบปฏิบัติการ MacOS X 10.12 Sierra



ภาพแสดงระบบ System Restore ของระบบปฏิบัติการ Microsoft Windows

1.2 การออกแบบ

ทางคณะผู้จัดทำได้ใช้ FUSE สำหรับการสร้าง File System บน User Space โดยใช้ระบบปฏิบัติการ Ubuntu Linux ในการพัฒนา และสำหรับการ compile source code จะใช้ Ninja ร่วมกับ mesonbuild โดยใช้ที่จัดเก็บข้อมูลอยู่ในระบบของไฟล์ ext2 ซึ่งมีอยู่ในระบบปฏิบัติการ Linux แล้ว

1.3 เงื่อนไขการออกแบบ

ฟังก์ชัน append มีเอาไว้เพื่อเรียกใช้ฟังก์ชัน add โดยรองรับการทำงานแบบหลายเทรดพร้อม ๆ กัน โดยจะมีคุณสมบัติดังนี้

- สามารถสร้าง ลบ เปลี่ยนชื่อ Directory ได้

- สามารถเปิด File สร้าง File ใหม่ และเขียน File ได้

- File หรือ Directory ที่ถูกเขียนลงบน Storage นั้นสามารถถูกเรียกกลับขึ้นมาดู (อ่าน) ได้

- สามารถทำ version ของไฟล์ที่มีอยู่แล้วได้โดยอัตโนมัติ นั่นคือ ผู้ใช้สามารถกลับไปใช้ไฟล์ version ก่อนหน้าได้ โดยในแต่ละ directory จะมี directory พิเศษชื่อ **archive** ที่ถูกสร้างขึ้นโดยอัตโนมัติจาก vCowFS โดย

ภายใน archive จะมีรายชื่อของไฟล์ตามด้วยเลข version ของไฟล์ใน directory ก่อนหน้านี้ โดยเวอร์ชันใหม่ของไฟล์จะถูกสร้างขึ้นโดยอัตโนมัติถ้าไฟล์นั้นถูกเขียนห่างจากครั้งสุดท้ายเกินระยะเวลาที่กำหนด ยกตัวอย่าง เช่น ถ้ากำหนด auto-snapshot delay เป็น 10 นาที ไฟล์เวอร์ชันใหม่จะถูกสร้างขึ้นโดยอัตโนมัติถ้ามีการเขียน/แก้ไฟล์ห่างจากเดิมเกิน 10 นาที

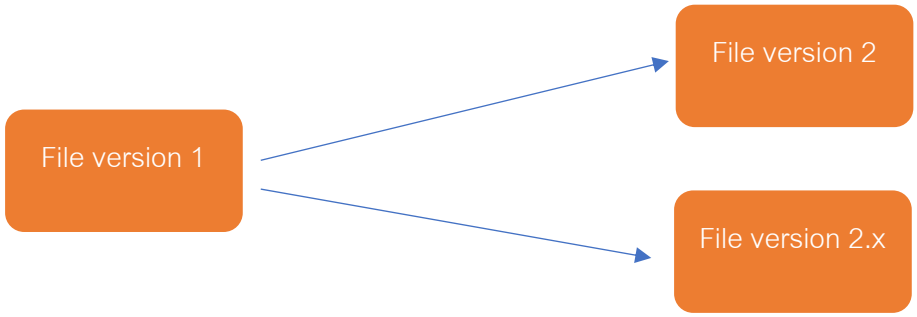
Chapter 2 Program Design

2.1 หลักการทำงานของ vCowFS

หลักการทำงานของ Copy-on-Write ที่ใช้ในการพัฒนาในครั้งนี้ มาจากแนวความคิดที่ว่า หากเราได้ทำการสร้างไฟล์ขึ้นมา 1 ไฟล์ เราจะทราบ Logical Address ของไฟล์ดังกล่าวจากนั้นจะนำไปเก็บไว้ใน Physical Memory โดยมีลักษณะเป็น Block โดยมี Physical Address เป็นตัวบอกตำแหน่ง ซึ่งการใช้วิธี Copy-on-Write ทำให้เราไม่ต้องอ้างตำแหน่งของ Physical Address ในการเข้าถึง และสามารถใช้ Share Data ใน Physical Memory ได้

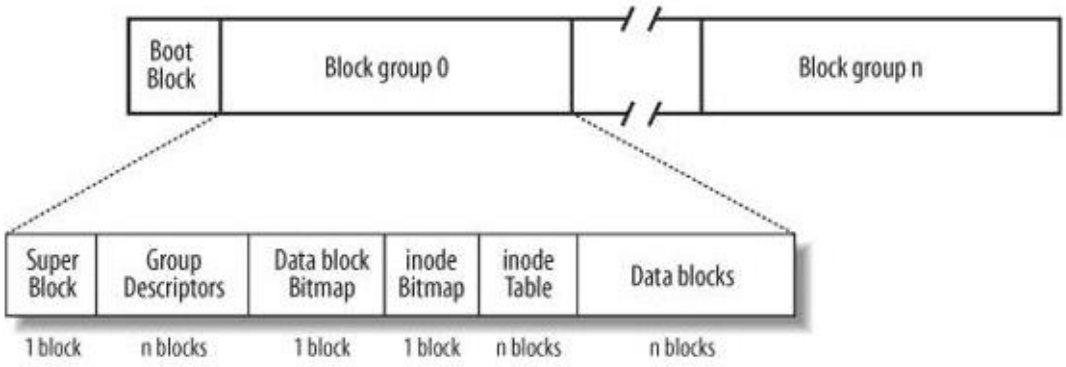
การเข้าถึงไฟล์ของ Copy-on-Write จะใช้วิธีการแบบ Journalist ซึ่งเป็นการเขียนไฟล์ขึ้นมาใหม่โดยไม่เขียนทับไฟล์เดิม (Overwriting) และจะมีการเก็บ Log สำหรับไฟล์ที่มีการเปลี่ยนแปลงแต่ละครั้ง

การทำ Versioning จะเป็นการสร้างไฟล์ขึ้นมาเป็น Version ใหม่เมื่อมีการแก้ไขไฟล์เดิม คล้ายๆกับ Backup ทำให้สามาถเรียกใช้ไฟล์ในเวอร์ชั่นก่อนหน้าได้

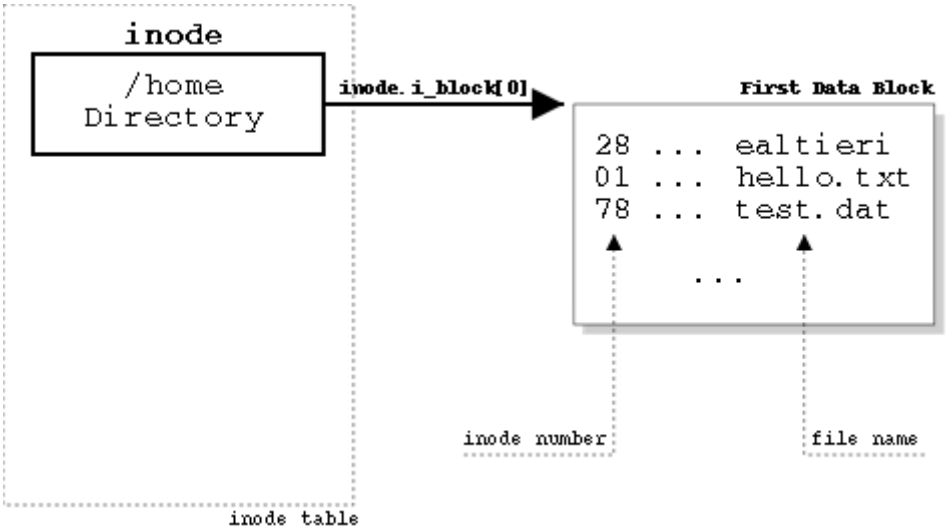


การออกแบบระบบไฟล์ในครั้งนี้ ได้อ้างอิงโครงสร้างบางส่วนมาจาก EXT2 ซึ่งเก็บไฟล์เป็น block ของข้อมูล โดยเมื่อ Mount ออกมาแล้วจะมีโครงสร้างของ block ดังภาพ

- IMG file ที่สร้างมีขนาด 1MB
- แต่ละ Block มีขนาด 4KB รวมแล้วมีจำนวนทั้งหมด 256 blocks
- Superblock เป็นส่วนที่ระบุข้อมูลพื้นฐานของ storage ที่ทำการ Mount ออกมาว่ามีขนาดเท่าไหร่ โดยจะใช้พื้นที่ทั้งหมด 1 block ในที่นี้จะมีขนาด 4KB
- Group Descriptors เก็บ address ของ block bitmap และ inode bitmap ของ IMG file ที่สร้างขึ้น กำหนดให้มีขนาด 1 block (4KB)
- Inode bitmap (ทำหน้าที่ระบุสถานะของ inodes ว่าว่างอยู่หรือไม่) และ data bitmap (ทำหน้าที่ระบุสถานะของ blocks ว่าว่างอยู่หรือไม่) สามารถเก็บข้อมูลได้ bitmap ละ 1 block
- Inode table ทำหน้าที่เก็บ address ของแต่ละ inode (inode คือข้อมูลเบื้องต้นของไฟล์ 1 ไฟล์ เช่นชื่อไฟล์อะไร metadata) กำหนดให้มีขนาด 1 block (4KB)
- Data blocks คือส่วนที่เหลือจากการทำส่วนอื่นๆ ทั้งหมด (251 blocks ขนาด block ละ 4KB)



มีการเก็บข้อมูลของ directory โดยจะระบุที่อยู่ของ inode ที่เก็บข้อมูล และมีการระบุชื่อไฟล์พร้อมทั้ง inode number ของไฟล์นั้นๆ เพื่อทำการค้นหา data blocks ที่เก็บไฟล์นั้นเป็นลำดับต่อไป



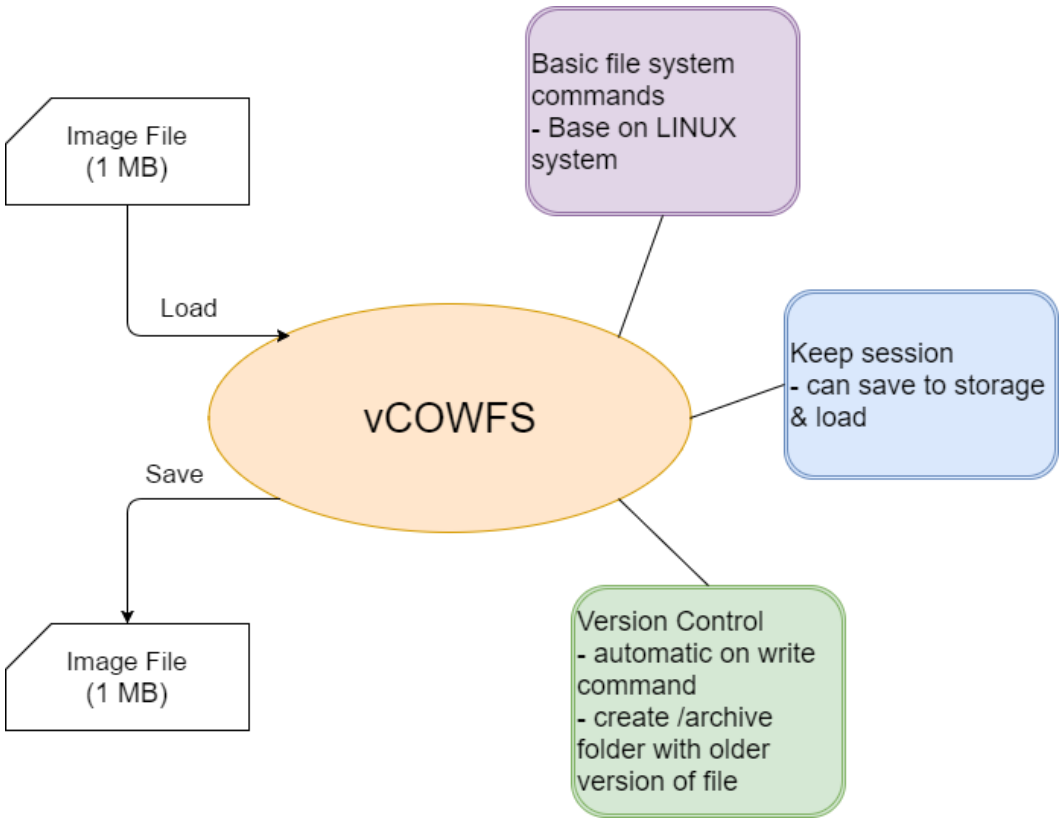
Inode metadata

NAME	หน้าที่
Mode	Permission ของไฟล์ ได้แก่ read, write และ execute
UID	ID ของ user ที่เข้าถึง
Size	ขนาดไฟล์
Create	เวลาที่ไฟล์ถูกสร้าง
Edit	เวลาที่ไฟล์ถูกแก้ไขล่าสุด
Delete	เวลาที่ถูกลบ
Group	Group ID
Blocks	Pointer ชี้ไปยัง blocks ที่เก็บไฟล์
Version	เวอร์ชั่นของไฟล์
Type	เป็น file หรือ directory

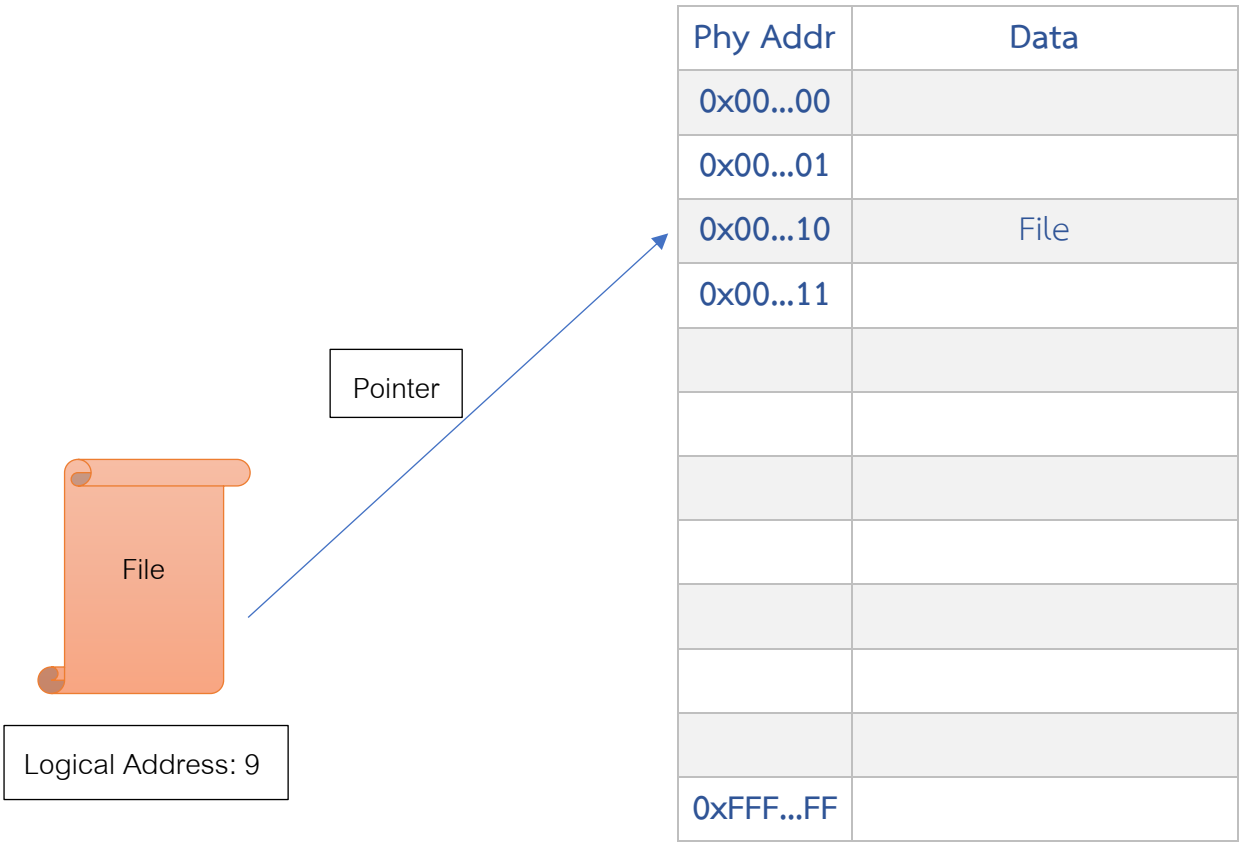
2.2 Feature การใช้งานหลักของ File system

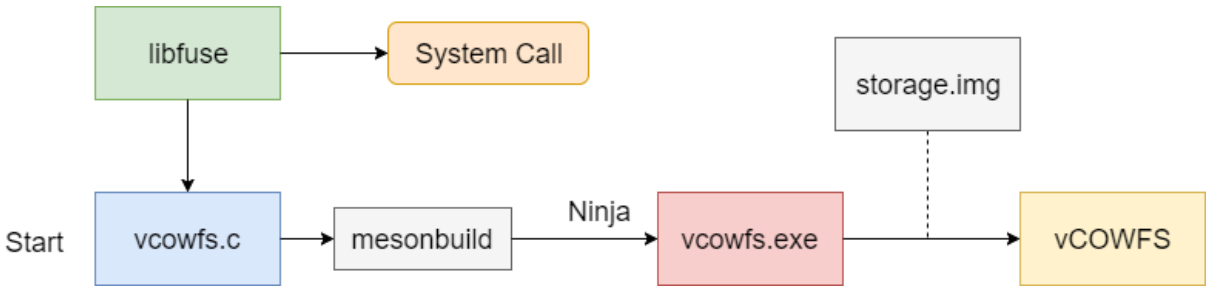
การออกแบบ File System ในครั้งนี้ไฟล์ที่สร้างขึ้นมาจะต้อง Mount ผ่านไฟล์ image แบบ ext2 ซึ่งมีอยู่ในระบบปฏิบัติการสำหรับจำลองเป็นที่เก็บข้อมูลเป็น Visual Storage ซึ่งใช้สำหรับเก็บไฟล์ที่พัฒนาขึ้นมา

การเข้าถึงไฟล์ของ File System จะเข้าถึงโดยอาศัยการ Mapping ระหว่าง Logical Address ซึ่งใช้อ้างอิงไฟล์ที่ถูกเขียนขึ้นมา กับ Physical Address ซึ่งเป็นตำแหน่งของ Storage โดยการเก็บข้อมูลใน Visual Disk ที่สร้างขึ้นมานั้นใช้รูปแบบของ Data Structure แบบ Array โดยการแบ่งที่อยู่ออกเป็น Block ของข้อมูลแล้วใช้ Physical Address กำกับเป็น Index ซึ่งจะเข้าถึง Address ดังกล่าวได้จากการใช้ Pointer ชี้ไปดังนี้



ภาพรวมการทำงานของ vCOWFS

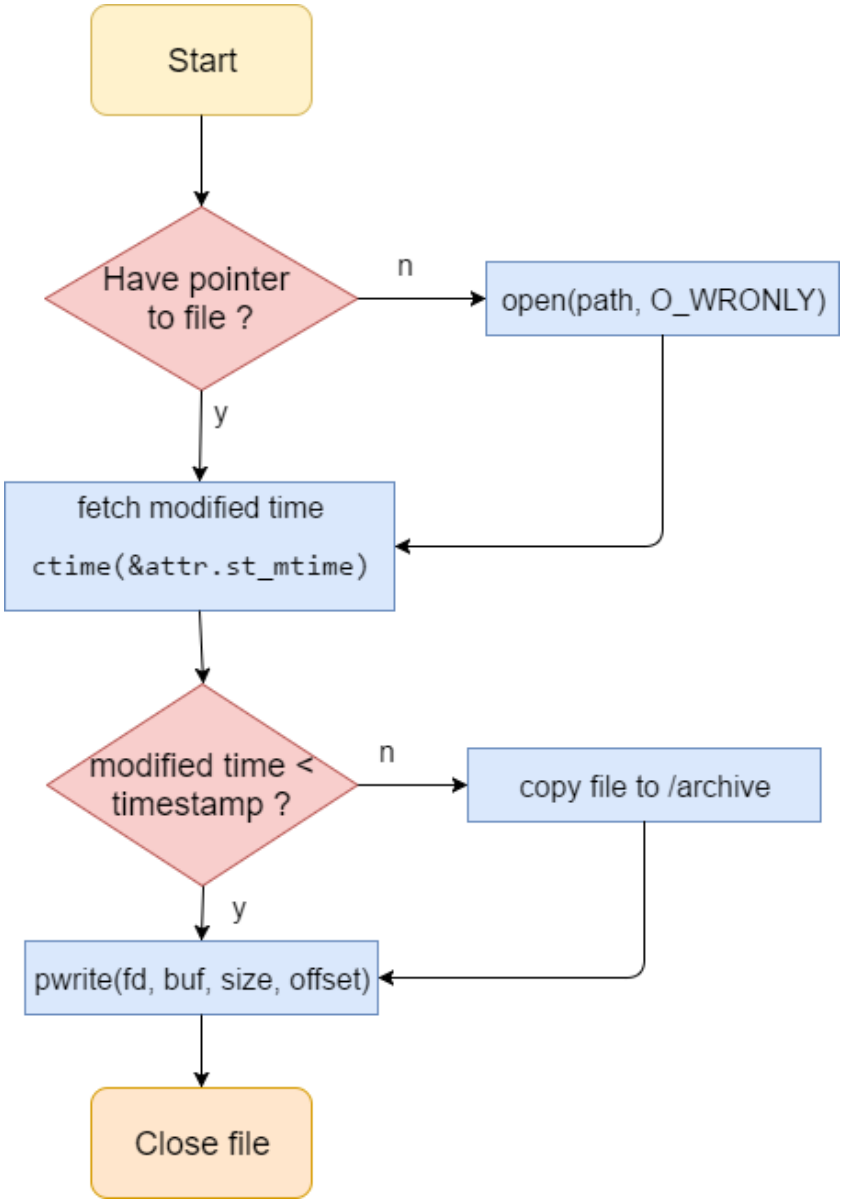




ภาพแสดงขั้นตอนการดำเนินงานสร้าง vCOWFS

2.3 กระบวนการสร้างเวอร์ชั่นใหม่ของแต่ละไฟล์

เริ่มจากสร้างไฟล์เริ่มต้นขึ้นมาก่อน ไฟล์ดังกล่าวจะมีการสำรองข้อมูลโดยการสร้างไฟล์ Version ใหม่ขึ้นมาตามช่วงเวลาที่เรากำหนด (Copy) โดยไฟล์มีคุณสมบัติที่จะสามารถเปิดไฟล์เดิมที่ทีอยู่ก่อนหน้าขึ้นมาได้ และสามารถทำสำเนาจากไฟล์เดิม จากนั้นจะมีการสร้าง Directory ใหม่ขึ้นมาโดยใช้ชื่อว่า archive ทุกครั้งที่มีการสร้างไฟล์ Version ใหม่ขึ้นมา (หรือปรับเปลี่ยนได้ เพราะไฟล์มีคุณสมบัติในการลบหรือเปลี่ยนชื่อ Directory) โดยมีหลักการทำงานดัง Flowchart นี้



Archive Design

- ใน **Write function** ดำเนินขั้นตอนดังต่อไปนี้

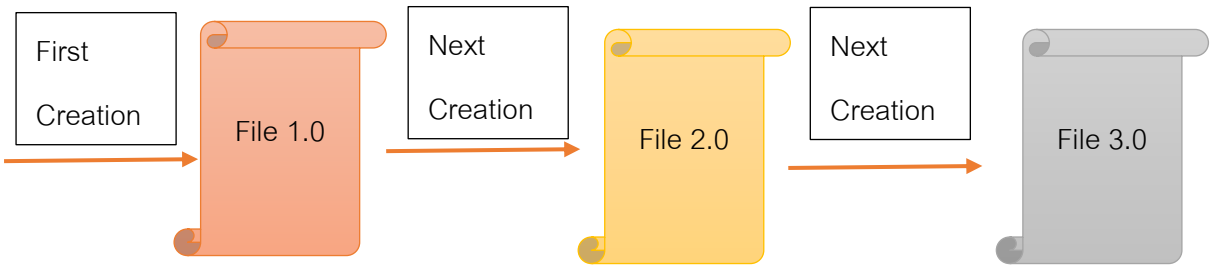
1. หาค่าเวลาล่าสุดที่ไฟล์ถูกแก้ไข ด้วยฟังก์ชั่น ctime()
2. ถ้าเลยเวลาจากที่ parameter รับมา ให้ write ไฟล์ลงใน archive ก่อน
3. Write version ล่าสุดใน directory ปัจจุบัน

- ใน **Read function** ดำเนินขั้นตอนดังต่อไปนี้

1. ทำการตรวจสอบไฟล์เดิมว่ามีอยู่หรือไม่ เป็น Version อะไร
2. ถ้าพบไฟล์เดิมว่ามีอยู่ ก็จะทำการเปิดไฟล์เพื่อเข้าถึงข้อมูลภายใน โดยฟังก์ชั่น Read จะมีการรับค่า Address ของไฟล์เข้ามา จากนั้นจะมี Pointer Variable อยู่ซึ่งจะมีหน้าที่ชี้ไปยัง Address ที่ไฟล์อยู่ จากนั้นจะดึงเข้าสู่ Buffer
3. ตำแหน่งของ Address นั้นจะทำการเข้าถึงแบบสุ่ม ซึ่งจะทำการส่งกลับมาเก็บไว้เป็น log เพื่อให้สามารถดูไฟล์เก่าย้อนหลังได้
4. การหา Directory ของไฟล์นั้นจะใช้ *path ในการชี้ไป

Version Control

ในการสร้างไฟล์ใหม่แต่ละครั้งจะมีการสร้าง Version ใหม่ให้กับไฟล์ที่ได้ทำการสร้างขึ้นมา โดยทุก Version ที่ได้จะถูกควบคุมโดย Version Control โดยง่ายที่สุดคือทุกครั้งที่มีการสร้างไฟล์ใหม่จะให้ Version เพิ่มขึ้นครั้งละ 1 เพื่อทำการบอกว่าไฟล์ใดเป็นไฟล์ล่าสุด



Chapter 3 Source Code

```c
/*
  Operating System KMITL
    Group 3 Assignment 2
    vCOWFS via FUSE
 */


#define FUSE_USE_VERSION 30

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#ifdef linux
/* For pread()/pwrite()/utimensat() */
#define _XOPEN_SOURCE 700
#endif

#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <dirent.h>
#include <errno.h>
#include <sys/time.h>

int watch,old_timestamp;
static const char *image_path = "mnt";

static void *vcow_init(struct fuse_conn_info *conn,  //Initialize the filesystem. This function can often be left unimplemented
               struct fuse_config *cfg)
{
    (void) conn;
    cfg->use_ino = 1;

    /* Pick up changes from lower filesystem right away. This is
       also necessary for better hardlink support. When the kernel
       calls the unlink() handler, it does not know the inode of
       the to-be-removed entry and can therefore not invalidate
       the cache of the associated sinode - resulting in an
       incorrect st_nlink value being reported for any remaining
       hardlinks to this inode. */
    cfg->entry_timeout = 0;
    cfg->attr_timeout = 0;
    cfg->negative_timeout = 0;

    return NULL;
}

static int vcow_getattr(const char *path, struct stat *stbuf, //Return file attributes
                struct fuse_file_info *fi)
{
    (void) fi;
    int res;

    res = lstat(path, stbuf);
    if (res == -1)
        return -errno;

    return 0;
}

static int vcow_access(const char *path, int mask)
{
    int res;

    res = access(path, mask);
    if (res == -1)
        return -errno;

```

```
73      return 0;
74  }
75
76  static int vcow_readlink(const char *path, char *buf, size_t size)
77  {
78      int res;
79
80      res = readlink(path, buf, size - 1);
81      if (res == -1)
82          return -errno;
83
84      buf[res] = '\0';
85      return 0;
86  }
87
88
89  static int vcow_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
90                  off_t offset, struct fuse_file_info *fi,
91                  enum fuse_readdir_flags flags)
92  {
93      DIR *dp;
94      struct dirent *de;
95
96      (void) offset;
97      (void) fi;
98      (void) flags;
99
100     //printf( "--> Trying to read %s, %u, %u\n", path, 55555, 55555 );
101
102     dp = opendir(path); //Open a directory for reading.
103     if (dp == NULL)
104         return -errno;
105
106     while ((de = readdir(dp)) != NULL) {
107         struct stat st;
108         memset(&st, 0, sizeof(st));
109         st.st_ino = de->d_ino;
110         st.st_mode = de->d_type << 12;
111         if (filler(buf, de->d_name, &st, 0, 0))
112             break;
113     }
114
115     closedir(dp);
116     return 0;
117 }
118
119 static int vcow_mknod(const char *path, mode_t mode, dev_t rdev) //Make a special
    (device) file
120 {
121     int res;
122
123     /* On Linux this could just be 'mknod(path, mode, rdev)' but this
124        is more portable */
125     if (S_ISREG(mode)) {
126         res = open(path, O_CREAT | O_EXCL | O_WRONLY, mode);
127         if (res >= 0)
128             res = close(res);
129     } else if (S_ISFIFO(mode))
130         res = mkfifo(path, mode);
131     else
132         res = mknod(path, mode, rdev);
133     if (res == -1)
134         return -errno;
135
136     return 0;
137 }
138
139 static int vcow_mkdir(const char *path, mode_t mode) //Create a directory with the
    given name.
140 {
141     int res;
142     char str[] = "Hello";
143     //path = strcat(path,str);
```

```
144         res = mkdir(path, mode);
145         if (res == -1)
146             return -errno;
147 
148         return 0;
149     }
150 
151     static int vcow_unlink(const char *path)//Remove (delete) the given file,
152     {
153         int res;
154 
155         res = unlink(path);
156         if (res == -1)
157             return -errno;
158 
159         return 0;
160     }
161 
162     static int vcow_rmdir(const char *path) //Remove the given directory.
163     {
164         int res;
165 
166         res = rmdir(path);
167         if (res == -1)
168             return -errno;
169 
170         return 0;
171     }
172 
173     static int vcow_symlink(const char *from, const char *to) //Create a symbolic link
    named "from" which, when evaluated, will lead to "to".
174     {
175         int res;
176 
177         res = symlink(from, to);
178         if (res == -1)
179             return -errno;
180 
181         return 0;
182     }
183 
184     static int vcow_rename(const char *from, const char *to, unsigned int flags)
    //Rename the file, directory, or other object "from" to the target "to".
185     {
186         int res;
187 
188         if (flags)
189             return -EINVAL;
190 
191         res = rename(from, to);
192         if (res == -1)
193             return -errno;
194 
195         return 0;
196     }
197 
198     static int vcow_link(const char *from, const char *to)//Create a hard link between
    "from" and "to". Hard links aren't required for a working filesystem, and many
    successful filesystems don't support them.
199     {
200         int res;
201 
202         res = link(from, to);
203         if (res == -1)
204             return -errno;
205 
206         return 0;
207     }
208 
209     static int vcow_chmod(const char *path, mode_t mode, //Change the mode (permissions)
    of the given object to the given new permissions.
210                 struct fuse_file_info *fi)
211     {
```

```
212         (void) fi;
213         int res;
214
215         res = chmod(path, mode);
216         if (res == -1)
217             return -errno;
218
219         return 0;
220     }
221
222     static int vcow_chown(const char *path, uid_t uid, gid_t gid, //Change the given
    object's owner and group to the provided values.
223                 struct fuse_file_info *fi)
224     {
225         (void) fi;
226         int res;
227
228         res = lchown(path, uid, gid);
229         if (res == -1)
230             return -errno;
231
232         return 0;
233     }
234
235     static int vcow_truncate(const char *path, off_t size, //Truncate or extend the
    given file so that it is precisely size bytes long.
236                 struct fuse_file_info *fi)
237     {
238         int res;
239
240         if (fi != NULL)
241             res = ftruncate(fi->fh, size);
242         else
243             res = truncate(path, size);
244         if (res == -1)
245             return -errno;
246
247         return 0;
248     }
249
250     static int vcow_create(const char *path, mode_t mode,
251                   struct fuse_file_info *fi)
252     {
253         int res;
254
255         res = open(path, fi->flags, mode);
256         if (res == -1)
257             return -errno;
258
259         fi->fh = res;
260         return 0;
261     }
262
263     static int vcow_open(const char *path, struct fuse_file_info *fi)
264     {
265         int res;
266
267         res = open(path, fi->flags);
268         if (res == -1)
269             return -errno;
270
271         fi->fh = res;
272         return 0;
273     }
274
275     static int vcow_read(const char *path, char *buf, size_t size, off_t offset,
276                 struct fuse_file_info *fi)
277     {
278         int fd;
279         int res;
280
281         if(fi == NULL)
282             fd = open(path, O_RDONLY);
```

```
283         else
284             fd = fi->fh;
285
286         if (fd == -1)
287             return -errno;
288
289         res = pread(fd, buf, size, offset);
290         if (res == -1)
291             res = -errno;
292
293         if(fi == NULL)
294             close(fd);
295         return res;
296     }
297
298     static int vcow_write(const char *path, const char *buf, size_t size,
299                  off_t offset, struct fuse_file_info *fi)
300     {
301         int fd;
302         int res;
303
304         (void) fi;
305         if(fi == NULL)
306             fd = open(path, O_WRONLY);
307         else
308             fd = fi->fh;
309
310         if (fd == -1)
311             return -errno;
312
313         res = pwrite(fd, buf, size, offset);
314         if (res == -1)
315             res = -errno;
316
317         if(fi == NULL)
318             close(fd);
319         return res;
320     }
321
322     static int vcow_statfs(const char *path, struct statvfs *stbuf)
323     {
324         int res;
325
326         res = statvfs(path, stbuf);
327         if (res == -1)
328             return -errno;
329
330         return 0;
331     }
332
333     static int vcow_release(const char *path, struct fuse_file_info *fi)
334     {
335         (void) path;
336         close(fi->fh);
337         return 0;
338     }
339
340     static int vcow_fsync(const char *path, int isdatasync,
341                  struct fuse_file_info *fi)
342     {
343         /* Just a stub.  This method is optional and can safely be left
344            unimplemented */
345
346         (void) path;
347         (void) isdatasync;
348         (void) fi;
349         return 0;
350     }
351
352     static struct fuse_operations vcow_oper = {
353         .getattr    = vcow_getattr,
354         .mknod      = vcow_mknod,
355       .mkdir    = vcow_mkdir,
```

```
356       .unlink     = vcow_unlink,
357       .rmdir      = vcow_rmdir,
358       .rename     = vcow_rename,
359       .chmod      = vcow_chmod,
360       .chown      = vcow_chown,
361       .truncate   = vcow_truncate,
362       .open       = vcow_open,
363       .read       = vcow_read,
364       .write      = vcow_write,
365       .release    = vcow_release,
366           //.opendir  = vcow_opendir,
367       .readdir    = vcow_readdir,
368       //.releasedir = vcow_releasedir,
369       .fsync      = vcow_fsync
370           //.fsyncdir     = vcow_fsyncdir
371   };
372
373   int main(int argc, char *argv[])
374   {
375
376       char mount_path[300];
377       char *param_temp[2];
378     //
379     //    param_temp[1] = argv[2];
380     //    param_temp[0] = argv[0];
381     //
382     // // sprintf(mount_path, "mount %s %s", argv[1], argv[2]);
383     //  system(mount_path);
384     // //printf("%s\n",mount_path);
385     //
386     // return fuse_main(2,param_temp,&vcow_oper,NULL);
387
388     //Normal Mount
389       if(argc == 5 && !strcmp(argv[3], "-t"))
390       {
391           param_temp[0] = argv[0];
392           param_temp[1] = argv[2];
393
394           //sprintf(mount_path, "mount %s %s", argv[1], image_path);
395       sprintf(mount_path, "mount %s %s", argv[1], argv[2]);
396       system(mount_path);
397
398       old_timestamp = time(NULL);
399           watch = atoi(argv[4]);
400
401           fuse_main(2, param_temp, &vcow_oper, NULL);
402       }
403       else {
404           printf("ARGUMENT ERROR, vCOWFS must follow with this format\n");
405           printf("./vCOWFS <Image File> <Mount Point> -t <Auto-snapshot Delay
          (seconds)>\n");
406       }
407   }
```

Function Description

static void *vcow_init(struct fuse_conn_info *connstruct fuse_config *cfg)

เป็นฟังก์ชั่นสำหรับ Set ค่าเริ่มต้นโดยจะกำหนดค่าเริ่มต้น Timeout เป็น 0

static int vcow_getattr(const char *path, struct stat *stbuf, struct fuse_file_info *fi)

เป็นฟังก์ชั่นสำหรับส่งค่า File Attribute

static int vcow_access(const char *path, int mask)

เป็นฟังก์ชั่นสำหรับเข้าถึงไฟล์โดยจะรับ Directory ของไฟล์เข้ามา

static int vcow_readlink(const char *path, char *buf, size_t size)

เป็นฟังก์ชั่นสำหรับตรวจสอบว่ามีไฟล์เดิมอยู่ใน Buffer หรือไม่

static int vcow_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi, enum fuse_readdir_flags flags)

เป็นฟังก์ชั่นที่ Read Function ใช้ในการเข้าถึงไฟล์โดยผ่าน Directory Path ที่ถูกส่งมา พร้อมกับค่าต่างๆที่เกี่ยวข้องกับไฟล์

static int vcow_mkdir(const char *path, mode_t mode)

เป็นฟังก์ชั่นสำหรับสร้าง Directory

static int vcow_unlink(const char *path)

เป็นฟังก์ชั่นสำหรับลบ ไฟล์ที่อยู่ใน Directory

static int vcow_rmdir(const char *path)

เป็นฟังก์ชั่นสำหรับลบ Directory

static int vcow_rename(const char *from, const char *to, unsigned int flags)

เป็นฟังก์ชั่นสำหรับเปลี่ยนชื่อ Directory, File name

static int vcow_chmod(const char *path, mode_t mode, struct fuse_file_info *fi)

เป็นฟังก์ชั่นสำหรับเปลี่ยน Permission

static int vcow_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi)

เป็นฟังก์ชั่นสำหรับ Read

static int vcow_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi)

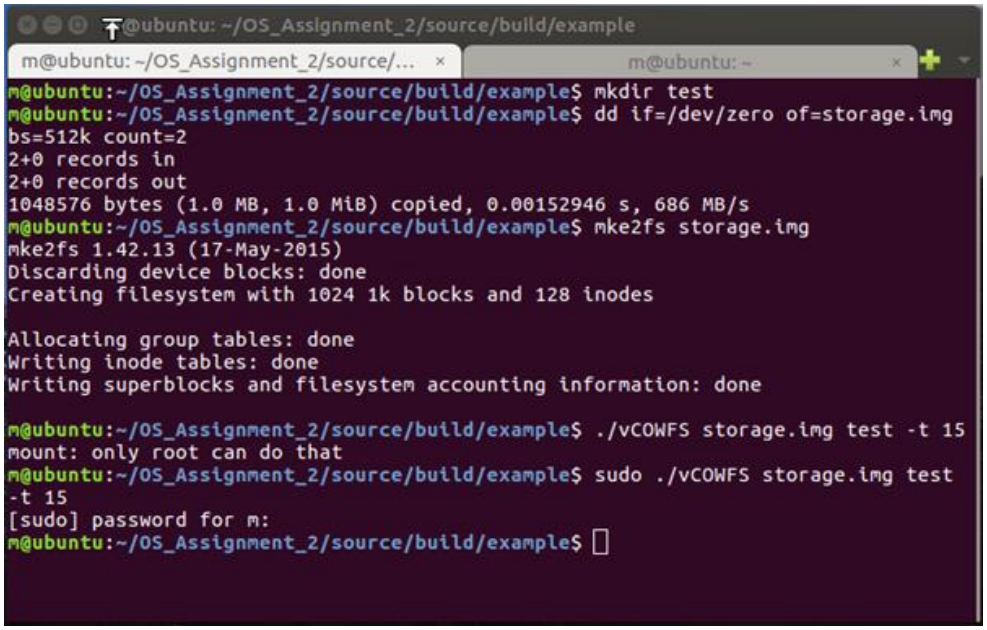
เป็นฟังก์ชั่นสำหรับ Write

Chapter 4 Program Ability

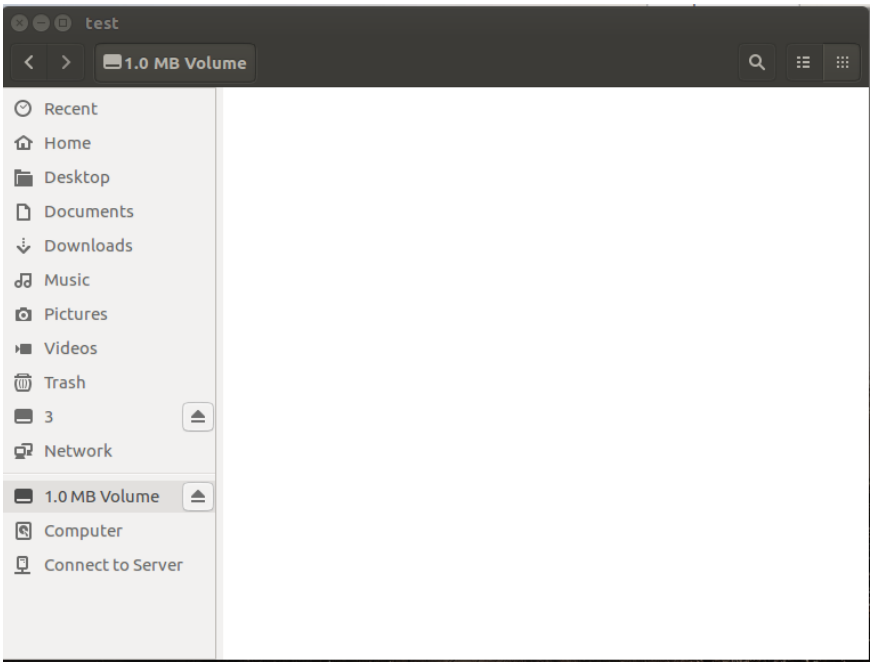
ID	Ability	Check
1	open/create/close/delete/truncate a file	✓
2	create/remove/rename/delete a directory	✓
3	chmod/chown a file or directory	✓
4	successfully listing files and directories with correct name, size, date, owner	✗
5	successfully listing files and directories with correct last modified date and time	✗
6	successfully reading a file	✓
7	successfully writing and syncing a file	✓
8	successfully auto-versioning a file	✗
9	successfully retrieve an old-version of the file	✗
10	successfully retrieve the file on re-mounting	✓

Chapter 5 Solution

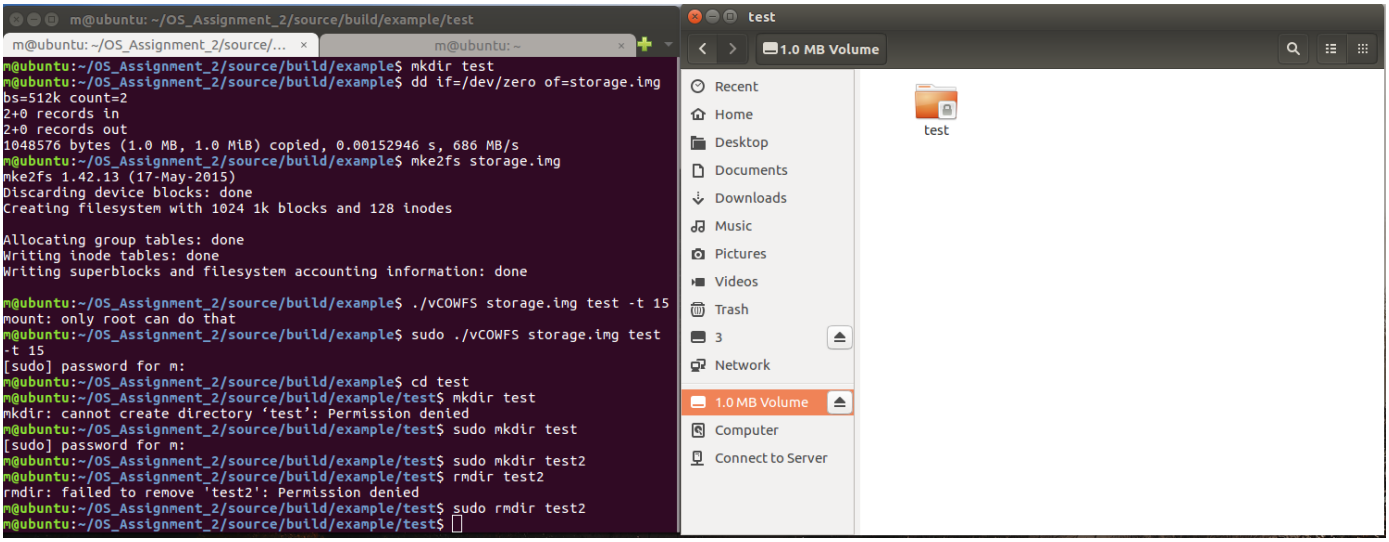
- สร้าง folder test สำหรับ mount
- สร้าง image file ชื่อ storage.img
- ทำการรันโปรแกรมโดย จะรับ parameter คือ path ของ Image File (storage.img) และ mount point (test) ที่ต้องการใช้



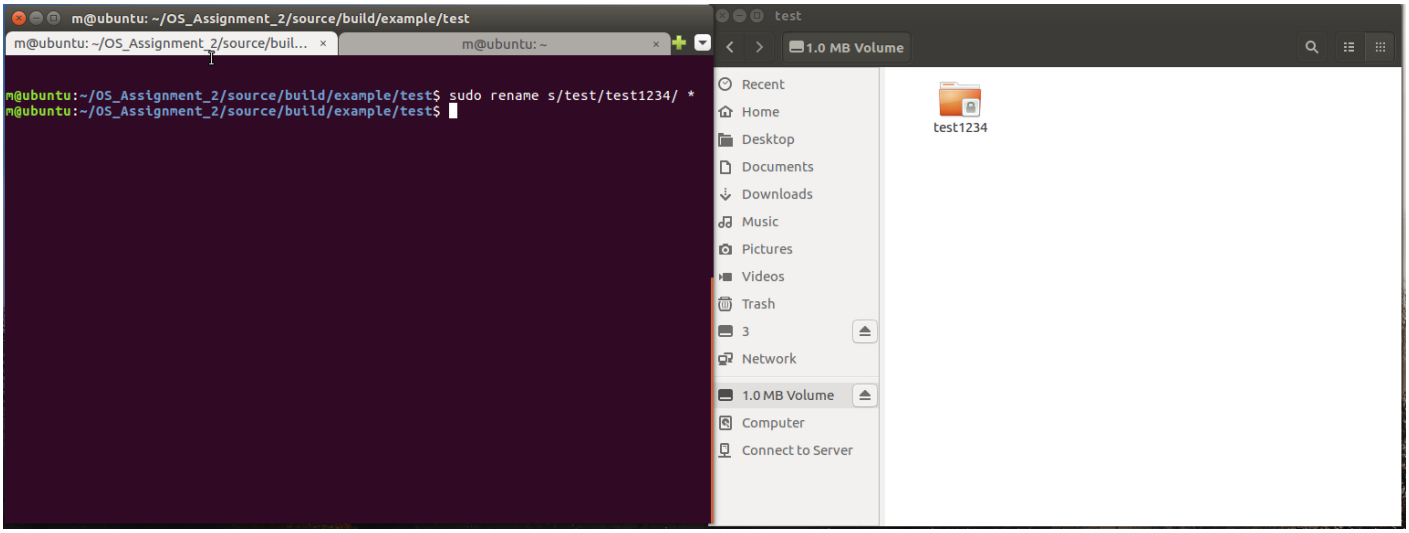
จะได้ storage ขนาด 1.0 MB ตามที่เราสร้างไว้



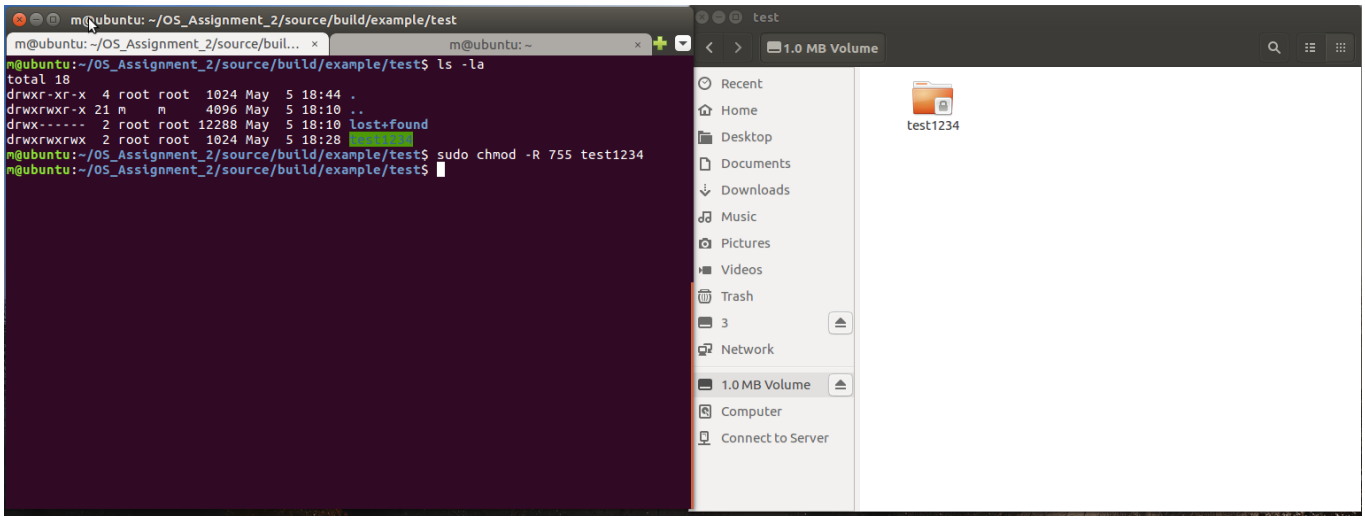
สามารถสร้าง folder และลบ folder ได้



สามารถเปลี่ยนชื่อได้



สามารถเปลี่ยนสิทธิการเข้าถึงได้



สามารถสร้างไฟล์ได้