檔案及目錄

認識檔案的基本屬性

檔案的基本屬性（ls -alh)

```
-rw-------  1 shiwulo shiwulo 2.5K Jan 28 09:59 .ICEauthority
drwx------  3 shiwulo shiwulo 4.0K Dec 29 12:44 .local
-rw-rw-r--  1 shiwulo shiwulo    0 Jan 28 10:15 ltrace_dropbox
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44 Music
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44 .parallels
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44 Pictures
-rw-r--r--  1 shiwulo shiwulo  675 Dec 29 05:41 .profile
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44 Public
lrwxrwxrwx  1 shiwulo shiwulo   57 Feb 23 09:44 sp -> /home/shiwulo/Desktop/Parallels Shared Folders/Dropbox/sp
```

為什麼要用“ ls -alh”而不是“ ls”

- `a`：代表印出所有的檔案，包含隱藏檔
  - 檔名使用「`.`」開頭的視為隱藏檔
  - 「`.`」代表當前的目錄
  - 「`..`」代表上一層目錄
  - 課堂小作業：回到上一層目錄
- `l`：代表印出多一點的訊息
- `h`：將檔案大小以人類易懂的方式顯示，例如：`1000` 會印成 `1K`

檔案的基本屬性

```
-rw-r--r--  2 shiwulo shiwulo 8.8K Dec 29 05:41
examples.desktop -rw-r--r--  1 shiwulo shiwulo  675 Dec 29
05:41 .profile
```

- 紅色的第 1 個字代表目錄（d）或檔案（-）捷徑（l）
- 2-9 總共九個字，代表各種人（擁有者、群組、其他人）的讀寫的權利， rwx 分別代表讀取、寫入、執行
- 因此 - rw- r-- r--，代表「這是檔案」「擁有者可讀寫」「群組可以讀」「其他人可以讀」

檔案的基本屬性

```
-rw-r--r--  2 shiwulo shiwulo 8.8K Dec 29 05:41
examples.desktop -rw-r--r--  1 shiwulo shiwulo  675 Dec 29
05:41 .profile
```

- 2 與 1 分別代表有多少「連結」（有一點像是 Windows 的捷徑）連到這個檔案（後面會介紹 Linux 的「捷徑」）
- 第一個 shiwulo 代表「擁有者」是 shiwulo，第二個 shiwulo 代表這個檔案的「群組」是 shiwulo

檔案的基本屬性

```
-rw-r--r--  2 shiwulo shiwulo 8.8K Dec 29 05:41
examples.desktop -rw-r--r--  1 shiwulo shiwulo  675 Dec 29
05:41 .profile
```

- 「8.8K」「675」分別代表檔案大小
- 「 Dec 29 05:41 」及「 Dec 29 05:41 」是上一次的「存取時間」
- 「 examples.desktop 」及「 .profile 」則是檔名，請注意當檔名的第一個字母是小數點「.」時，該檔案是隱藏檔，如果「ls」所下的指令未包含「a」，就不會顯示隱藏檔

rwx 在目錄上的權限含義

- 在傳統 UNIX 系統上，目錄是一個「特別的檔案」，這個檔案記錄了「其他檔案的檔名、屬性」及「對應的檔案分配表（inode）」

- 「r」代表可以可以讀取該目錄，例如：印出該目錄的內容
- 「w」代表可以修改這個目錄，例如：「新增、刪除、更名、移動」一個檔案
- 「x」代表可以進入該目錄

檔案的特殊屬性

process 有三個 `id`

- effective user id (euid)
  - 真正用來判斷權限的 id
- real user id (ruid)
  - 該 process 的 owner 的 user id
- saved-user-id
  - 當 euid 改變時，將舊的 euid 存放在 saved-user-id

set-uid

- #include <unistd.h>
- int setuid(uid_t uid);
- int setgid(gid_t gid);

- 將 euid 設定為 uid，如果設定正確回傳 0，失敗回傳 -1，失敗的原因紀錄在 errno
- 一個 set-uid 的程式，執行時， euid 會等於這個程式的 owner
- set-uid 的程式可以靠 setuid() 於：「程式 owner」及「執行者間作切換」

或許是最有名的應用： passwd

- **$** ls /etc/shadow -alh
- -rw-r----- 1 root shadow 1.3K Apr 10 14:43 /etc/shadow
- shadow 內部存放「密碼相關資訊」只有 root 可以改 shadow 這個檔案。那麼一般人怎樣修改密碼呢?

- **$** ls /usr/bin/passwd -alh
- -rwsr-xr-x 1 root root 53K May 17 2017 /usr/bin/passwd
- 使用 passwd 這支程式，執行這支程式時會暫時變成 root （這個執行檔的 owner ）的權限

- strace -c passwd

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         2         2 setgroups
  0.00    0.000000           0         1           setresuid
  0.00    0.000000           0         2           setresgid
```

- setresuid 是 setuid 的進階版

lab: 製造一個超級 ls

- 複製 /bin/ls
  cp /bin/ls ./sls
- 變更擁有者為 super user
  sudo chown root sls
- 加入 set user id bit
  sudo chmod +s ./sls
- 測試
  - ./sls /root 「可以讀取該目錄」
  - ./sls /lost+found 「可以讀取該目錄」
  - ls /root 「沒有『 set-uid 』的 ls ，無法讀取該目錄」
- 課堂作業：仿造「 sls 」的做法，製造一個能讀取任何檔案的「超級 less 」

Changing UIDs and GIDs

- 如果是超級使用者（super user, root）呼叫 setuid(uid) 那麼 real user ID, effective user ID 和 saved-user-ID 都會等於 uid。
  - 因此 super user 可以變身為任何人，但是一旦變身以後，就無法變身回去了
- 如果是一般使用者（normal user）那麼呼叫 setuid(uid) 以後
  - 參數「uid」必須等於 real user ID 或 saved-user-id，否則會產生錯誤
  - setuid 會將 effective user ID 設定為 real user ID 或 saved-user-id

課堂小作業

- 寫一隻程式，可以變成任何人
- 提示： system(“bin/bash”);
- 提示：記得要對你的執行檔設定 set-uid 權限

Change （變身為任意人）

```
1.  #include <stdlib.h>
2.  #include <unistd.h>
3.  #include <stdio.h>

4.  int main(int argc, char**argv) {
5.      int uid, ret;
6.
7.      sscanf(argv[1], "%d", &uid);
8.      ret = setuid(0);
9.      printf("ret=%d", ret);
10.     setuid(uid);
11.     system("/bin/bash");
12.     return 0;
13. }
```

```
sudo chown root ./change
sudo chmod +s ./change
sudo adduser guest1
/* 假設 guest1 的 user ID 是 1001*/
./change 1001
$ whoami
guest1
```

setreuid and setregid

- #include <unistd.h>
- int setreuid(uid_t ruid, uid_t euid);
- int setregid(gid_t rgid, gid_t egid);

設定 real user ID 和 effective user ID 。這個函數通常用來交換 (swap) real user ID 和 effective user ID 讓 set-uid 的程式暫時性的切換到 real uid 。正確回傳 0 ，否則回傳 -1 。

seteuid

- #include <unistd.h>
- int seteuid(uid_t uid);
- int setegid(gid_t gid);

- 只改變 effective user ID 。錯誤回傳 -1 否則回傳 0

getuid()

- #include <unistd.h>
- #include <sys/types.h>
- uid_t getuid(void);
- uid_t geteuid(void);

getresuid()

- **#define _GNU_SOURCE**
- #include <unistd.h>
- int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid);
- int getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid);

- 拿到 real user ID、 effective user ID 及 saved-user-ID， 錯誤回傳 -1 否則回傳 0

sticky bit

設定一個目錄，該目錄只有目錄擁有者、檔案擁有者或超級使用者（super user， root）可以刪除檔案。

- 通常使用於暫存目錄，例如： /tmp
- 指令 chmod +t

- `ls / -alh`
- `drwxrwxrwt  1 root root  736 Apr 17 14:24 tmp`

小結

- 已經了解 Linux 基本的權限控制
- 知道在 Linux 中如何切換行程的身份

硬連結及軟連結

關於 hard link

- hard link 是讓目錄結構內，多個項目（可能是檔案，也可能是目錄）指向另一個項目（檔案或目錄）『在 Linux 中只可連向檔案』
- hard link 所指向的新路徑與舊路徑必須存在於同一個 partition
- 只有當 hard link 的數量變成 0 時，該檔案才會被真正的刪除
  - `-rw-r--r--  2 shiwulo shiwulo 8.8K Dec 29 05:41 examples.desktop`
- 一個檔案的多個 hard link 可以各自擁有自己的權限，因此透過 hard link 可以讓一個檔案擁有多個不同權限（後面我們會介紹更一般化的方法）

關於 soft link

- 是一個特別的檔案（類似於 Windows 的捷徑）連向某個檔案或目錄
- 就算我們擁有存取 softlink 的權限，我們還需要有使用該檔案的權限。
- softlink 可以跨過不同的 partition
- softlink 可以指向一個不存在的東西
- softlink 不會影響 link 的數量
  - `-rw-r--r--  2 shiwulo shiwulo 8.8K Dec 29 05:41 examples.desktop`
  - 在這個例子中，無論創建了多少 softlink 都不會改變「2」

hard link

- Linux 指令： ln （hard link），ln -s （soft link）

- #include <unistd.h>
- /*hard link*/
- int link(const char *oldpath, const char *newpath);
- /*soft link*/
- int symlink(const char *oldpath, const char *newpath);

link

- 撰寫一支程式，可以建立 hard link 及 soft link
  - 例子，建立 hard link： link -s source target
  - 例子，建立 soft link ： link source target

link

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  #include <errno.h>


4.  int main(int argc, char **argv)
5.  {
6.      int ret;
7.      printf("%d \n", argc);
8.      if (argc == 4) {/*softlink*/
9.          ret = symlink(argv[2], argv[3]);
10.         printf("%s, %s\n", argv[2], argv[3]);
11.
12.     if (ret != 0) perror("soft
13.         link:");
14.   } else {
15.     ret = link(argv[1], argv[2]);
16.     printf("%s, %s\n", argv[1],
17.       argv[2]);
18.     if (ret != 0) perror("hard
19.       link");
20.   }
21.   return 0;
22.}
```

執行結果

```
shiwulo@vm:~/sp/ch06$ ./a.out -s link.c l
4
link.c, l
shiwulo@vm:~/sp/ch06$ ./a.out link.c ll
3
link.c, ll
shiwulo@vm:~/sp/ch06$ ls l* -alh
lrwxrwxrwx 1 shiwulo shiwulo   6  二  17 13:56 l -> link.c
-rw-rw-r-- 2 shiwulo shiwulo 406  二  17 13:54 link.c
-rw-rw-r-- 2 shiwulo shiwulo 406  二  17 13:54 ll
```

測試連向一個目錄

```
$ ln ch06 chxx
ln: ch06: hard link not allowed for directory
```

小結

- 了解到 soft link 相當於 windows 的「捷徑」
- 了解 hard link 的意涵，只有移除所有的「 hard link 」及「原檔案」後，該檔案才會真正的消失
- 要知道 soft link 和 hard link 在權限上的不同

操作目錄及檔案屬性

mkdir

- Linux 指令： mkdir

- #include <sys/stat.h>
- #include <sys/types.h>
- int mkdir(const char *pathname, mode_t mode);

- 製造一個目錄於 pathname，目錄權限是 mode & ~umask &0777
- 「umask」後面會介紹，基本上它是用來限制最高權限

rmdir

- Linux 指令： rmdir

- #include <unistd.h>
- int rmdir(const char *pathname);

- 刪除目錄 pathname

chdir

- Linux 指令： cd

- #include <unistd.h>
- int chdir(const char *path);
- int fchdir(int fd);

- 改變目前的工作目錄

課堂作業

- 將目前的「工作目錄」更改為 path 或者 fd 所指向的目錄
- 在 Linux 中， cd 是內建指令（ internal ，因此無法使用 strace 觀看 cd 如何實作）
- 可以使用 system(“bash”);

mycd （這是一個爛例子，因為 cd 不可能用外部指令實現）

```
1. #include <unistd.h>
2. #include <stdlib.h>

3. int main(int argc, char**argv) {
4.     chdir(argv[1]);
5.     system("bash");
6.     return 0;
7. }
```

getcwd

- Linux 指令： pwd

- #include <unistd.h>
- char *getcwd(char *buf, size_t size);

- getcwd 會將目前的「絕對路徑」寫入到 buf，這個 buf 大小為 size，如果 buf 太小，那麼 Linux 就無法將路徑完整地寫出到 buf
- size 大小應該為 PATH_MAX，請注意： PATH_MAX 可能是執行時期決定的變數，因此應該用 malloc()

get_current_dir_name
比 getcwd 好用的函數

- **get_current_dir_name**() will **malloc**(3) an array big enough to hold the absolute pathname of the current working directory.
- The caller should **free**(3) the returned buffer.

課堂小作業

- 寫一隻程式 mypwd 可以印出現在的工作目錄

條列目錄裡所有的物件

列出目錄裡所有的東西

- Linux 指令： ls

- #include <dirent.h>
- DIR *opendir(const char *pathname);
- struct dirent *readdir(DIR *dp);
- int closedir(DIR *dp);

dirent 結構

```
1.  struct dirent {
2.      ino_t          d_ino;       /* inode number */
3.      off_t          d_off;       /* not an offset; see NOTES */
4.      unsigned short d_reclen;    /* length of this record */
5.      unsigned char  d_type;      /* type of file; not supported
6.                                     by all filesystem types */
7.      char           d_name[256]; /* filename */
8.  };
```

在這個資料結構中，最常用到的是 d_name ，可以藉由 d_name 及 stat() 拿到這個檔案的所有屬性

dir

```
1.    #include <stdio.h>
2.    #include <dirent.h>
3.    int main(int argc, char **argv)
4.    {
5.        DIR* dir;
6.        struct dirent* ent;
7.        dir = opendir(argv[1]);
8.        ent = readdir(dir);
9.        while(ent!=NULL) {
10.           printf("%s\n",  ent->d_name);
11.           ent = readdir(dir);
12.       }
13.       closedir(dir);
14.       return 0;
15.   }
```

執行結果

```
$ ./dir ./
.
..
chmod
chmod.c
dir
dir.c
dir2
dir2.c
dir3
dir3.c
link.c
lnk
```

listDir.c

```
1.    int level = 0;


2.    void printName(char* type, char* name) {
3.        printf("%s", type);
4.        for (int i=0; i < level; i++)
5.            printf("  ");
6.        if (strcmp("d", type)==0)
7.            printf("+");
8.        else
9.            printf("|");
10.       printf("%s\n", name);
11.   }
```

listDir.c

```
1.  void listDir(char* pathName)
2.  {
3.      level++;
4.      DIR* curDir = opendir(pathName);
5.      assert(curDir!=NULL);
6.      char* newPathName = (char*)malloc(PATH_MAX);
7.      struct dirent entry;
8.      struct dirent* result;
9.      int ret;
10.     ret = readdir_r(curDir, &entry, &result);
11.     while(result != NULL) {
12.         if (strcmp(entry.d_name, ".") == 0 ||
13.                 strcmp(entry.d_name, "..") == 0) {
14.             ret = readdir_r(curDir, &entry, &result);
15.             assert(ret == 0);
16.             continue;
17.         }
18.
```

```
1.          assert(ret == 0);
2.          if (entry.d_type == DT_LNK)
3.              printName("l", entry.d_name);
4.          if (entry.d_type == DT_REG)
5.              printName("f", entry.d_name);
6.          if (entry.d_type == DT_DIR) {
7.                  printName("d", entry.d_name);
8.                  sprintf(newPathName, "%s/%s", pathName,
9.                          entry.d_name);
10.                 printf("%s\n", newPathName);
11.                 listDir(newPathName);
12.             }
13.         ret = readdir_r(curDir, &entry, &result);
14.         assert(ret == 0);
15.     }
16.     closedir(curDir);
17.     level--;
18. }
19. int main(int argc, char** argv) {
20.         listDir(argv[1]);
21. }
```

執行結果

```
f  |2018- 系統程式期中考（筆試部分） .docx
d  +2018-sp-midterm
../2018-sp-midterm
d    +.vscode
../2018-sp-midterm/.vscode
f      |launch.json
f      |settings.json
f      |c_cpp_properties.json
f    |gettime
f    |listDir
```

小結

- 了解在在傳統 Unix-like 系統中，目錄的構造如同檔案
  - 但實際上現在的目錄可能是 B-tree 或者是 hash 構造
- 雖然目錄如同一個檔案，但打開目錄需要使用特別的函數
- 了解為什麼 get_current_dir_name 比 getcwd 要來得好，並且了解 get_current_dir_name 必須搭配 free 使用
- 複習了遞迴的使用方式

利用 stat 讀取檔案屬性

更進階版的 dir

- 目前的 dir 程式只能列印出檔案名稱，但無法知道這個檔案的屬性
- 如果要知道檔案的屬性，必須使用下一頁所列的三個函數

更進階版的 dir

```
1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <unistd.h>

4. int stat(const char *path, struct stat *buf);
5. int fstat(int fd, struct stat *buf);
6. int lstat(const char *path, struct stat *buf);
```

- stat 會將資料寫入 buf， lstat 會檢視「soft link」本身，而 fstat 的傳入值是 file descriptor
- 下一頁說明 struct stat

struct stat

```
1.  struct stat {
2.      dev_t     st_dev;     /* ID of device containing file */
3.      ino_t     st_ino;     /* inode number */
4.      mode_t    st_mode;    /* protection */
5.      nlink_t   st_nlink;   /* number of hard links */
6.      uid_t     st_uid;     /* user ID of owner */
7.      gid_t     st_gid;     /* group ID of owner */
8.      dev_t     st_rdev;    /* device ID (if special file) */
9.      off_t     st_size;    /* total size, in bytes */
10.     blksize_t st_blksize; /* blocksize for filesystem I/O */
11.     blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
12.     time_t    st_atime;   /* time of last access */
13.     time_t    st_mtime;   /* time of last modification */
14.     time_t    st_ctime;   /* time of last status change */
15. };
```

struct stat 常用的欄位說明

```
1.  struct stat {
2.      mode_t    st_mode;    /*  檔案類型及檔案權限 */
3.      nlink_t   st_nlink;   /*  多少 hard link 指到這個檔案 */
4.      uid_t     st_uid;     /* owner 的 ID */
5.      gid_t     st_gid;     /* group 的 ID */
6.      dev_t     st_rdev;    /* device ID (if special file) */
7.      off_t     st_size;    /* total size, in bytes */
8.      time_t    st_atime;   /*  最後存取時間 */
9.      time_t    st_mtime;   /*  上次修改時間 */
10.     time_t    st_ctime;   /*  修改這個資料結構（meta-data）的時間 */
11. };
```

dir2

```
1. #include <stdio.h>
2. #include <dirent.h>
3. #include <stdio.h>
4. #include <dirent.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/stat.h>
8. #include <unistd.h>
```

dir2

```
1. #include <time.h>

2. int main(int argc, char **argv) {
3.     DIR* dir;
4.     struct dirent* ent;
5.     char* curDir = "./";
6.     char pathname[512];
7.     struct stat buf;
8.     int perm;
```

dir2

```
1.      char *time;
2.
3.      dir = opendir(argv[1]);
4.      ent = readdir(dir);
5.      while(ent!=NULL) {
6.          strcpy(pathname, curDir);
7.          strcat(pathname, ent->d_name);
8.          stat(pathname, &buf);
9.          perm = (buf.st_mode & \
10.             (S_IRWXU | S_IRWXG | S_IRWXO));
```

dir2

```
1.          time = ctime(&buf.st_atime);
2.          time[strlen(time)-1] = 0;
3.          printf("%o  %d  %d %8d %s %s\n",\
4.              perm, buf.st_uid, buf.st_gid,\
5.              (int)buf.st_size, time, ent->d_name);
6.          ent = readdir(dir);
7.      }
8.      closedir(dir); return 0;
9. }
```

執行結果

```
$ ./dir2 ./
755  1000  1000       340 Sat Mar 12 06:42:29 2016 .
755  1000  1000       680 Fri Mar 11 16:35:03 2016 ..
775  1000  1000      8663 Fri Mar 11 17:10:20 2016 chmod
664  1000  1000       305 Fri Mar 11 13:52:03 2016
chmod.c
775  1000  1000      8709 Sat Mar 12 06:42:25 2016 dir
664  1000  1000       261 Fri Mar 11 17:05:58 2016 dir.c
775  1000  1000      9092 Sat Mar 12 06:42:29 2016 dir2
664  1000  1000       762 Sat Mar 12 06:36:12 2016 dir2.c
664  1000  1000       298 Fri Mar 11 17:10:20 2016 link.c
664  1000  1000       298 Fri Mar 11 17:10:20 2016 lnk
```

更進階版的 dir

- 目前 dir 的輸出結果已經很像 ls -al
- 但使用者名稱和群組名稱都是數字，而非有意義的字串

- 使用下列二個函數

```
1. #include <pwd.h>
2. struct passwd *getpwuid(uid_t uid);
3. #include <grp.h>
4. struct group *getgrgid(gid_t gid);
```

dir3

- 將輸出函數換成新的輸出函數

```
1. printf("%o  %d  %d %8d %s %s\n", perm,\
2. buf.st_uid, buf.st_gid,\
3. (int)buf.st_size, time, ent->d_name);
```

```
4. printf("%o  %s  %s %8d %s %s\n", perm, \
5. getpwuid(buf.st_uid)->pw_name,
   getgrgid(buf.st_gid)->gr_name,\
6. (int)buf.st_size, time, ent->d_name);
```

執行結果

```
$ ./dir3 ./
755  shiwulo  shiwulo      408 Sat Mar 12 07:02:42 2016 .
755  shiwulo  shiwulo      680 Fri Mar 11 16:35:03 2016 ..
775  shiwulo  shiwulo     8663 Sat Mar 12 06:59:25 2016 chmod
664  shiwulo  shiwulo      305 Fri Mar 11 13:52:03 2016 chmod.c
775  shiwulo  shiwulo     8709 Sat Mar 12 06:59:25 2016 dir
664  shiwulo  shiwulo      261 Fri Mar 11 17:05:58 2016 dir.c
775  shiwulo  shiwulo     9092 Sat Mar 12 06:59:25 2016 dir2
664  shiwulo  shiwulo      762 Sat Mar 12 06:59:04 2016 dir2.c
775  shiwulo  shiwulo     9200 Sat Mar 12 07:02:42 2016 dir3
664  shiwulo  shiwulo      831 Sat Mar 12 07:02:30 2016 dir3.c
664  shiwulo  shiwulo      298 Sat Mar 12 06:59:25 2016 link.c
664  shiwulo  shiwulo      298 Sat Mar 12 06:59:25 2016 lnk
```

小結

- 本小結最主要的重點在於「了解如何取得檔案屬性」
- 了解 ctime, mtime, atime 的不同
  - mtime 、 atime 分別代表修改時間、存取時間
  - ctime ，修改 meta data 的時間，也就是修改 i-node 的時間， Linux 並未提供函數修改 ctime
- 了解如何走訪目錄結構

檔案的操作

檔案權限的遮罩 umask

- Linux 指令： umask

- #include <sys/types.h>
- #include <sys/stat.h>
- mode_t umask(mode_t mask);

- 使用 umask 可以設定「檔案遮罩」，所有新建立的檔案或者檔案權限的修改都會受到這個檔案遮罩的影響。 umask 的回傳值是「舊的 umask 的設定值」
- 例如：使用 umask 將檔案遮罩設定為「 002 」，那麼新建立的檔案的「其他人」就一定不會有「寫入權限」

課堂小作業

- 寫一支程式，該程式會在螢幕上印出目前 umask 的設定，並且不會改變 umask 的設定值

unlink

- Linux 指令： rm

- #include <stdio.h>
- int remove(const char *pathname);

- 刪除 pathname，在 Linux 中由於一個檔案可能被多個「路徑名」參照，因此「刪除 pathname」的實際功能是讓「參照數」少 1，如果參照數變為 0，系統就會真正刪除這個檔案 / 資料夾（資料夾的參考數都為 1）
- 要特別注意的是，如果正好有一個程式打開 (open) 了這個檔案，除非這個程式關閉 (close) 了這個檔案，否則這個檔案會以「隱形的方式」存在於檔案系統，這是避免暫存檔案佔據空間的重要技巧。

課堂小作業

- 寫一隻小程式，可以移除某個檔案或者資料夾，例如：
- myrm /path/something

rename()

- Linux 指令： rm

- #include <stdio.h>
- int rename(const char *old, const char *new);

- 將 old 名字改成 new 。

rename

```
1. #include <stdio.h>

2. int main(int argc, char **argv)
3. {
4.     rename(argv[1], argv[2]);
5.     return 0;
6. }
```

結果

```
$ ./rename tmp ../tmp
$ cd ..
$ ls tmp
tmp
```

chmod, chgrp, chowner

- Linux 指令 : chgrp, chown, chmod

```
#include <sys/types.h>
#include <unistd.h>
int chown(const char *path, uid_t owner, gid_t group);

#include <sys/stat.h>
int chmod(const char *path, mode_t mode);
int fchmodat(int fd, const char *path, mode_t mode, int flag);
```

課堂作業

- 自行用 man 這個指令查看這三個函數的用途

修改權限

- Linux 指令： chmod

- `#include <sys/stat.h>`
- `int chmod(const char *path, mode_t mode);`
- `int fchmod(int fd, mode_t mode);`

課堂作業

- 撰寫一支程式，可以改變檔案權限

chmod

```
1.  #include <stdio.h>
2.  #include <sys/stat.h>
3.  int main(int argc, char **argv)
4.  {
5.      int perm;
6.      int owner, grp, others;
7.      sscanf(argv[2], "%1d%1d%1d", &owner, &grp, &others);
8.      printf("permission = %d %d %d\n", owner, grp, others);
9.      perm = owner<<6 | grp<<3 | others;
10.     /*7<<6 == 111 000 000, 7<<3 == 111 000, 7 == 111*/
11.     chmod(argv[1], (mode_t)perm);
12.     return 0;
13. }
```

執行結果

```
$./chmod link.c 777
permission = 7 7 7
$ ls -alh link.c
-rwxrwxrwx 1 shiwulo shiwulo 298 Mar 11 14:23 link.c
```

小結

- 了解如何變更 UNIX （注意，不只是 Linux ）的檔案屬性變更方法

設定檔案的擴充權限

檔案的擴充屬性

- 傳統 Unix 只有三種屬性，分別是: owner、 group、 others，但有時候人員很複雜，需要做個細緻的規範
- 例如:
  - 老師可以針對作業進行任何修改
  - 助教：可以讀取任何作業
  - 學生：可以修改作業
  - 其他人：沒有權限
- 以上一個例子而言，權限就有四種，超出傳統 Unix 的權限

指令： getfacl, setfacl

- getfacl （acl： access control list）讀取一個檔案或目錄的擴充屬性
- setfacl，設定一個檔案或者目錄的擴充屬性

getfacl

```
$ getfacl examples.desktop    /* 對，就是這麼簡
單， getfacl 後面將路徑名 */
# file: examples.desktop   /* 檔名 */
# owner: shiwulo /* 檔案擁有者 */
# group: shiwulo /* 群組 */
user::rw- /*UNIX 預設屬性 rwx  rwx  rwx*/
group::r--
other::r--
```

setfacl

- `setfacl -m u:guest1:rx ./tmp`
  - `-m` 代表修改（`modify`）權限
  - `u：[使用者名稱]：[權限]  檔名`
- `setfacl -m g:guest1:rx ./tmp`
  - `-m` 代表修改（`modify`）權限
  - `g：[群組名稱]：[權限]  檔名`

執行結果

```
$setfacl -m u:guest1:rw ./tmp
$getfacl ./tmp
# file: tmp
# owner: shiwulo
# group: shiwulo
user::rw-    /* 沒有使用者（::）代表擁有者（owner）
user:guest1:rw-
group::rw-
mask::rw-
other::r--
```

執行結果

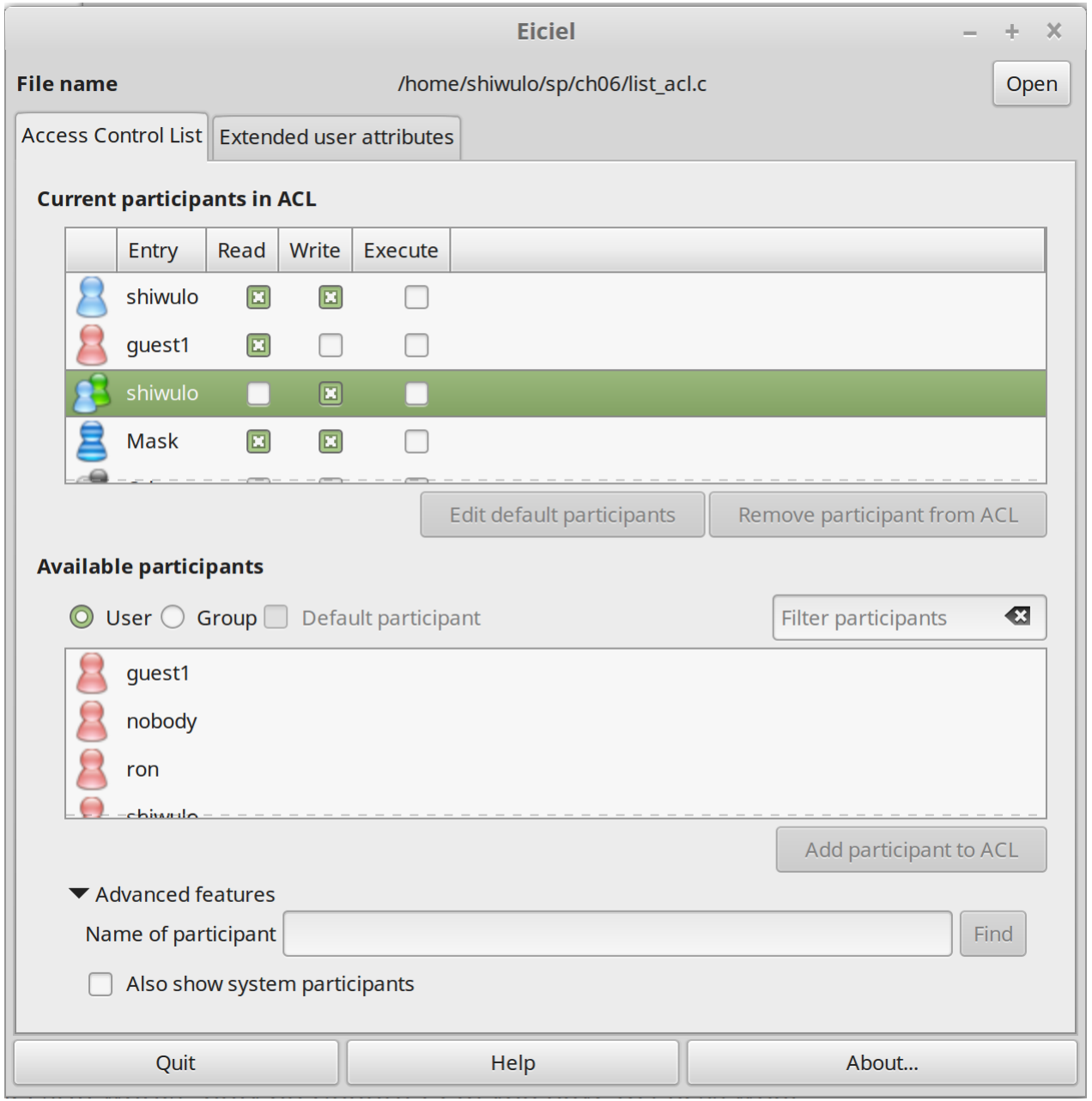
```
$setfacl -m g:guest1:rw ./tmp
$ getfacl ./tmp
# file: tmp
# owner: shiwulo
# group: shiwulo
user::rw-
user:guest1:rw-
group::rw-
group:guest1:rw-    /* 增加了新的群組 guset1，權限為可讀寫 */
mask::rw-
other::r--
```

ls 一下

```
$ls -lh
drwxrwxr-x  6 shiwulo shiwulo 4.0K Feb  1 12:25 _sp
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44
Templates
-rw-rw-r--+ 1 shiwulo shiwulo    0 Mar 15 08:14 tmp
/* 檔案屬性多了 + 代表這一個檔案有擴充屬性，要用 getfacl 才可以看到
完整的屬性 */
drwxr-xr-x  2 shiwulo shiwulo 4.0K Dec 29 12:44 Videos
```

GUI

```
sudo apt-get install eiciel
```



ACL functions

- `sudo apt-get install acl acl-dev`
- `acl_get_file(), acl_get_entry(), acl_get_tag_type(), acl_get_qualifier(), acl_get_permset(), acl_get_perm()`
- `sudo apt-get install clang`
- `clang -lacl list_acl.c`
- `gcc list_acl.c -lacl`

clang

Clang is designed to be highly compatible with GCC.

- Clang's command-line interface is similar to and shares many flags and options with GCC.
- Clang implements many GNU language extensions and enables them by default.

Clang's developers aim to reduce memory footprint and increase compilation speed compared to competing compilers, such as GCC.

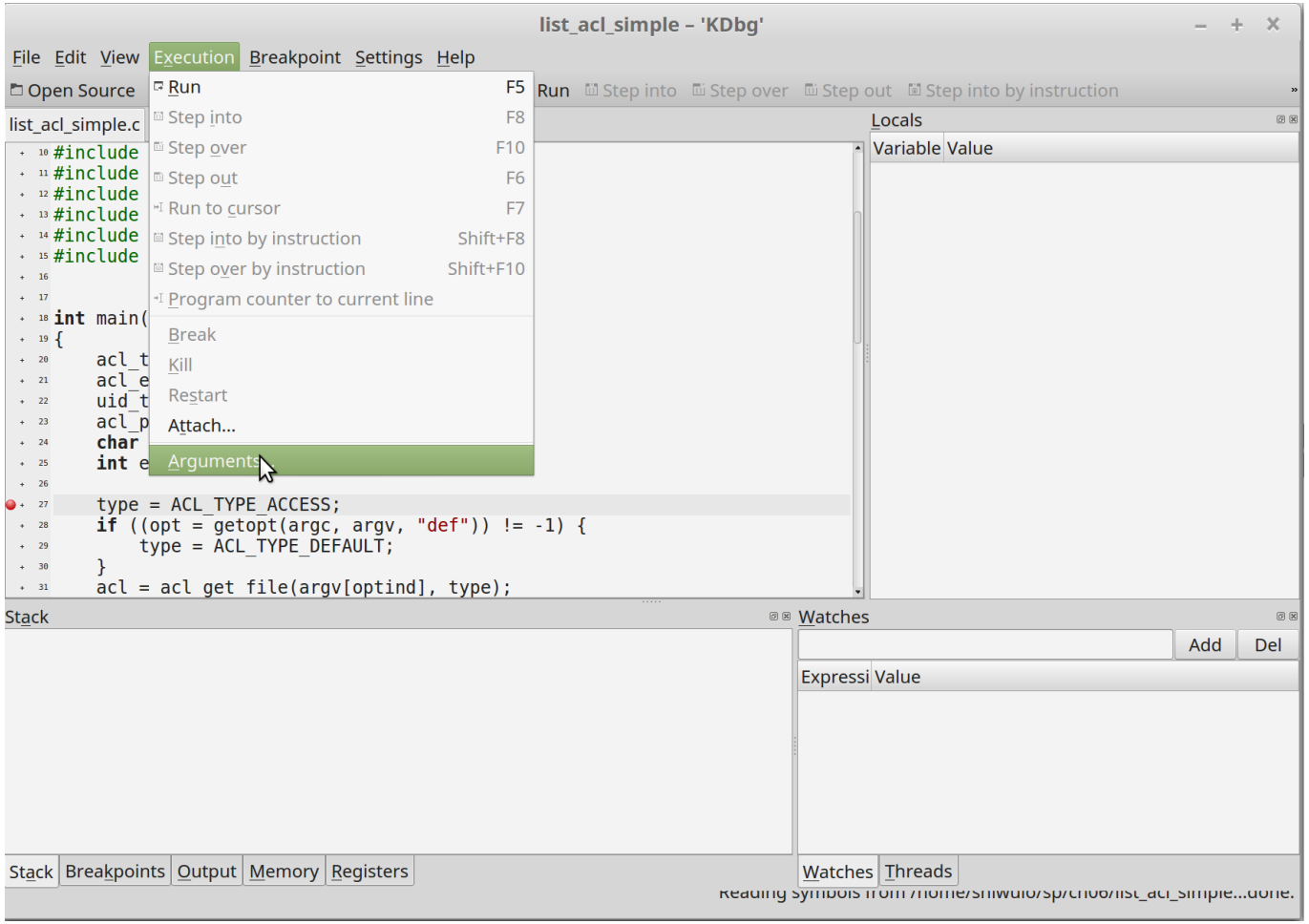
- More recent comparisons indicate that both compilers have evolved to increase their performance.
- 效能不分軒輊

position of gcc -l option

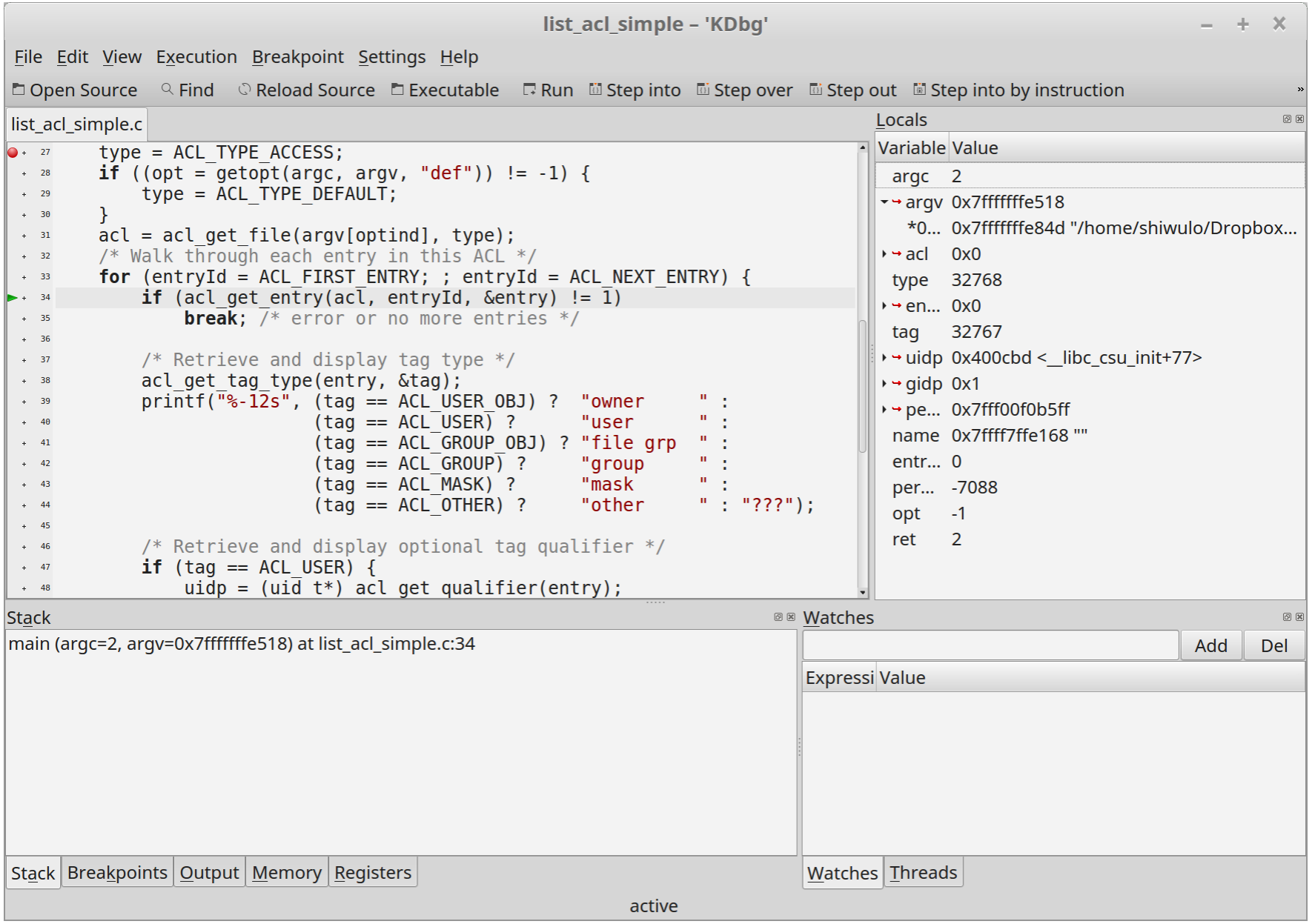
- It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified.
- Thus, foo.o -lz bar.o searches library z after file foo.o but before bar.o. If bar.o refers to functions in z, those functions may not be loaded.

Debugger-kdbg

$sudo apt-get install kdbg



kdbg


list_acl_simple – 'KDbg'
File Edit View Execution Breakpoint Settings Help
Open Source Find Reload Source Executable Run Step into Step over Step out Step into by instruction
list_acl_simple.c
27 type = ACL_TYPE_ACCESS;
28 if ((opt = getopt(argc, argv, "def")) != -1) {
29 type = ACL_TYPE_DEFAULT;
30 }
31 acl = acl_get_file(argv[optind], type);
32 /* Walk through each entry in this ACL */
33 for (entryId = ACL_FIRST_ENTRY; ; entryId = ACL_NEXT_ENTRY) {
34 if (acl_get_entry(acl, entryId, &entry) != 1)
35 break; /* error or no more entries */
36
37 /* Retrieve and display tag type */
38 acl_get_tag_type(entry, &tag);
39 printf("%-12s", (tag == ACL_USER_OBJ) ? "owner " :
40 (tag == ACL_USER) ? "user " :
41 (tag == ACL_GROUP_OBJ) ? "file grp " :
42 (tag == ACL_GROUP) ? "group " :
43 (tag == ACL_MASK) ? "mask " :
44 (tag == ACL_OTHER) ? "other " : "???");
45
46 /* Retrieve and display optional tag qualifier */
47 if (tag == ACL_USER) {
48 uidp = (uid_t*) acl_get_qualifier(entry);
Locals
Variable Value
argc 2
argv 0x7fffffffe518
*0... 0x7fffffffe84d "/home/shiwulo/Dropbox...
acl 0x0
type 32768
en... 0x0
tag 32767
uidp 0x400cbd <__libc_csu_init+77>
gidp 0x1
pe... 0x7fff00f0b5ff
name 0x7ffff7ffe168 ""
entr... 0
per... -7088
opt -1
ret 2
Stack
main (argc=2, argv=0x7fffffffe518) at list_acl_simple.c:34
Watches
Add Del
Expressi Value
Stack Breakpoints Output Memory Registers
Watches Threads
active

list_acl_simple

```
1.      acl = acl_get_file(argv[optind], type);
2.      /* Walk through each entry in this ACL */
3.      for (entryId = ACL_FIRST_ENTRY; ; entryId = ACL_NEXT_ENTRY) {
4.          if (acl_get_entry(acl, entryId, &entry) != 1)
5.              break; /* error or no more */
6.          acl_get_tag_type(entry, &tag); entries
7.          printf("%-12s", (tag == ACL_USER_OBJ) ?  "owner     " :
8.                           (tag == ACL_USER) ?     "user      " :
9.                           (tag == ACL_GROUP_OBJ) ? "file grp  " :
10.                          (tag == ACL_GROUP) ?    "group     " :
11.                          (tag == ACL_MASK) ?     "mask      " :
12.                          (tag == ACL_OTHER) ?    "other     " : "???");

13.         /* Retrieve and display optional tag qualifier */
14.         if (tag == ACL_USER) {
15.             uidp = (uid_t*) acl_get_qualifier(entry);
```

list_acl_simple

```
1.              printf("%-8s\t", getpwuid(*uidp)->pw_name);
2.          } else if (tag == ACL_GROUP) {
3.              gidp = (gid_t*) acl_get_qualifier(entry);
4.              printf("%-8s\t", getgrgid(*gidp)->gr_name);
5.          } else printf("\t\t");
6.          acl_get_permset(entry, &permset);
7.          permVal = acl_get_perm(permset, ACL_READ);
8.          printf("%c", (permVal == 1) ? 'r' : '-');
9.          permVal = acl_get_perm(permset, ACL_WRITE);
10.         printf("%c", (permVal == 1) ? 'w' : '-');
11.         permVal = acl_get_perm(permset, ACL_EXECUTE);
12.         printf("%c", (permVal == 1) ? 'x' : '-');
13.         printf("\n");
14.     }
15.     acl_free(acl);
```

list_acl_simple

```
$ setfacl -m u:guest1:rw ./list_acl.c
$ ./list_acl_simple ./list_acl.c
owner           rw-
user      guest1    rw-
file grp        rw-
mask            rw-
other           r-
```

strace list_acl list_acl.c

```
getxattr("./list_acl.c",
"system.posix_acl_access",
"...",
132) = 44
```

getxattr

- `#include <sys/types.h>`
- `#include <attr/xattr.h>`

- `ssize_t getxattr(const char *path, const char *name,`
- `                  void *value, size_t size);`
- `ssize_t lgetxattr(const char *path, const char *name,`
- `                   void *value, size_t size);`
- `ssize_t fgetxattr(int fd, const char *name,`
- `                   void *value, size_t size);`

- 課堂作業：上網查詢這三個函數的用法

listxattr

```
#include <sys/types.h>
#include <attr/xattr.h>

ssize_t listxattr (const char *path,
                   char *list, size_t size);
ssize_t llistxattr (const char *path,
                    char *list, size_t size);
ssize_t flistxattr (int filedes,
                    char *list, size_t size);
```

listattr.c

```
1. void print_value(char* pathname, char* attr) {
2.     char buf[4096];
3.     int total;
4.     total=getxattr(pathname, attr, buf, 4096);
5.     for (int i=0; i<total; i++)
6.         printf("%x", buf[i]);
7.     printf("\n\n");
8. }


9. int main (int argc, char** argv) {
10.    char *buf;
```

listattr.c

```
1.      int total, sum=0;
2.      char str[4096];
3.      buf = (char*)malloc(4096);
4.      total=listxattr(argv[1], NULL, 0);
5.      total=listxattr(argv[1], buf, total);
6.      for( ;sum !=total; ){
7.          sscanf(buf+sum, "%s", str);
8.          printf("xattr = %s\n", str);
9.          print_value(argv[1], str);
10.         sum += strlen(str)+1;
11.     }
12.}
```

listattr.c

```
$ ./listattr ./list_acl.c

xattr = user.com.dropbox.attrs

a14a10ffffff9bffffff8effffffd9fffffcfbffffffc4ffffffe9fffffff000000037ffffffc0101010fffff
f9affffffedffffff85ffffff8d3


xattr = user.com.dropbox.attributes

78ffffff9cffffffab564a29ffffffca2f48ffffffcaffffffaffffffff884ffffffffcbffffffcc49ffffffcd4c
ffffffff89ffffffcfffffffc94f4effffffcc51ffffffb252ffffffa856ffffffca4d4cffffffceffffffc8ffff
ffcc3ffffff8925ffffff96ffffff9414ffffff81ffffff8552124b12ffffff81c25ffffffff3ffffffaafffffff
8cffffffb4ffffffa0affffffed42ffffffcbffffffff0ffffffff0ffffffe0ffffffc4ffffffff424ffffff83fff
fffa47453ffffffaf54475b5bffffffa5ffffffdaffffffda5a0ffffffc9ffffff801c72


xattr = system.posix_acl_access

20001060ffffffffffffffffffffffffffffffff2060ffffffe93004060ffffffffffffffffffffffffffffffff
f10060ffffffffffffffffffffffffffffffff20040ffffffffffffffffffffffffffffffff
```

修改 listattr.c

```
total=getxattr(pathname, attr, buf, 4096);

nSize= getxattr(pathname, attr, buf, 0);
buf=malloc(nSize);
nSize=getxattr(pathname, attr, buf, nSize);
```

setxattr

```
#include <sys/types.h>
#include <sys/xattr.h>

int setxattr(const char *path, const char *name,
             const void *value, size_t size, int flags);
int lsetxattr(const char *path, const char *name,
              const void *value, size_t size, int flags);
int fsetxattr(int fd, const char *name,
              const void *value, size_t size, int flags);
```

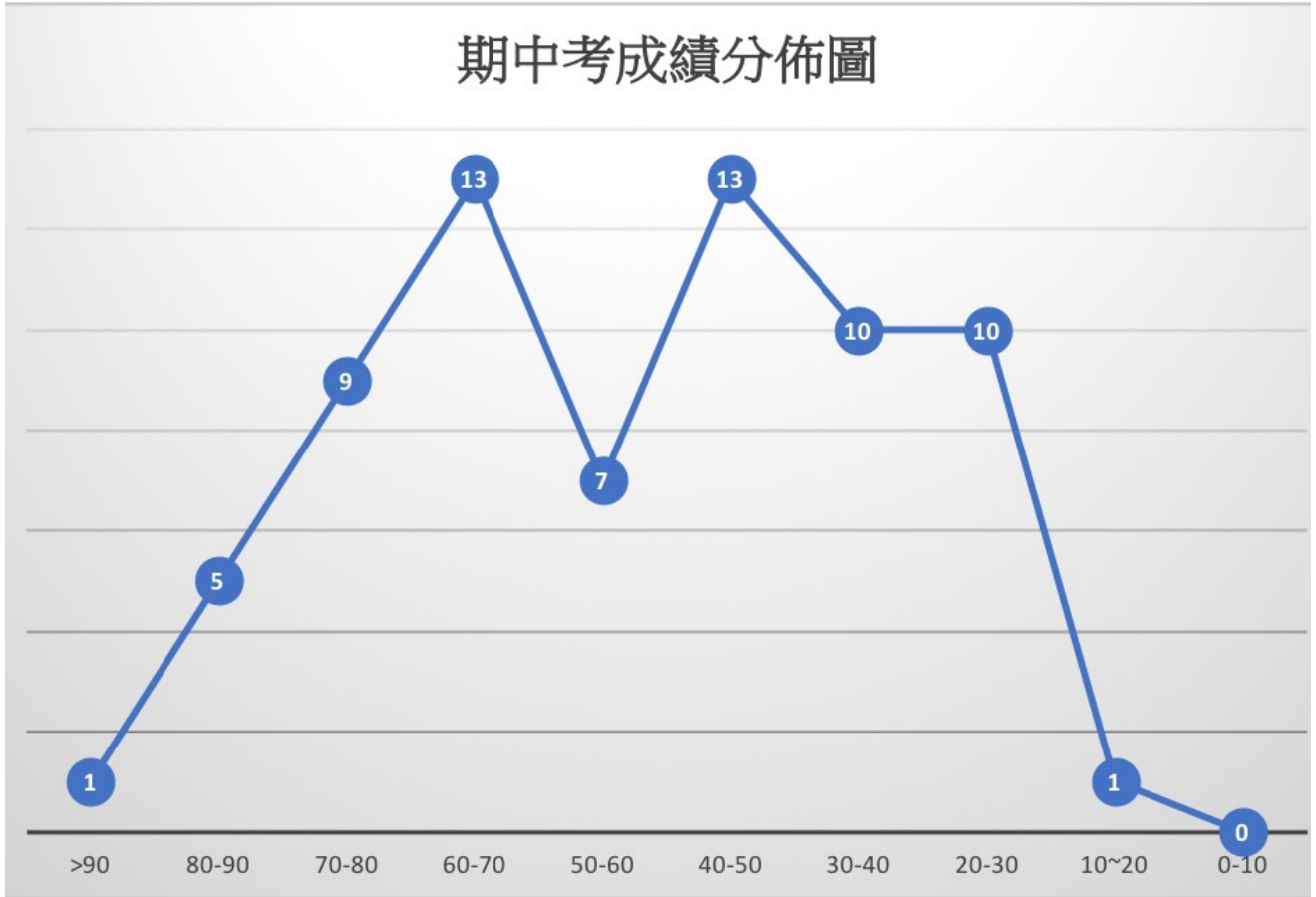
小結

- 本小節的重點是 Linux 的權限控制不只是 owner、 group、 others，還可以有更多的擴充權限
- 請注意，這些權限是附加於檔案系統上，因此檔案系統必須支援附加權限（如： ext4 就支援， FAT 不支援）

期中考成績

成績分布

- 成績查詢：www.ecourse.ccu.edu.tw
- 成績分布：如右圖
- 期中預警： 30 分以下
- 上機考成績不足 17 分的同學請下課後找我



監聽資料夾內的變動

試用 inotify

```
 $ sudo apt install inotify-tools
 $ inotifywait -m list_acl.c ..
Setting up watches.
Watches established.
../ OPEN,ISDIR ch06
../ ACCESS,ISDIR ch06
../ CLOSE_NOWRITE,CLOSE,ISDIR ch06
../ OPEN,ISDIR ch06
../ ACCESS,ISDIR ch06
../ CLOSE_NOWRITE,CLOSE,ISDIR ch06
../ OPEN,ISDIR ch06
../ ACCESS,ISDIR ch06
../ CLOSE_NOWRITE,CLOSE,ISDIR ch06
```

inotify

- `int inotify_init(void)`
  - 初始化一個 `inofiry` 的監聽，回傳值是 `file descriptor（fd）`
- `int inotify_add_watch(int fd, const char* pathname, int mask)`
  - 使用 `fd` 對一個檔案或者目錄（`pathname`）進行一些（`mask`）監聽，回傳 `watch descriptor（wd）`
- `int inotify_rm_watch(int fd, int wd)`
  - 移除 `inotify` 的一個監聽

- 設定好要監聽的物件（如：目錄、檔案）後，就可以用 `read` 來查看監聽的結果

inotify_event

```
struct inotify_event {
    int      wd;       /* Watch descriptor */
    uint32_t mask;     /* Mask describing event */
    uint32_t cookie;   /* Unique cookie associating related
                          events (for rename(2)) */
    uint32_t len;      /* Size of name field */
    char     name[];   /* Optional null-terminated name */
};
```

inotify.c

```
1.  int main(int argc, char **argv) {
2.      int fd, num ret, i;
3.      char* p;
    /*BUF_LEN (10 * (sizeof(struct inotify_event) + NAME_MAX + 1))*/
4.      char inotify_entity[BUF_LEN];
5.      fd = inotify_init();
6.      for (i=1; i< argc; i++) {
7.          ret=inotify_add_watch(fd, argv[i], IN_ALL_EVENTS);
8.          strcpy(wd[ret], argv[i]);
9.      }
10.     while(1) {
11.         num = read(fd, inotify_entity, BUF_LEN);
12.         for (p = inotify_entity; p < inotify_entity + num; ) {
13.             printInotifyEvent((struct inotify_event *) p);
14.             p+=sizeof(struct inotify_event) + ((struct inotify_event *)p)->len;
15.         }
16.     }
17. }
```

inotify.c

```
1.  void printInotifyEvent(struct inotify_event* event) {
2.      char buf[4096]="";
3.      sprintf(buf, "[%s] ", wd[event->wd]);
4.      strncat(buf, "{", 4096);
5.      if (event->mask & IN_ACCESS)         strncat(buf, "ACCESS, ", 4096);
6.      if (event->mask & IN_ATTRIB)         strncat(buf,"ATTRIB, ", 4096);
7.      if (event->mask & IN_CLOSE_WRITE)    strncat(buf,"CLOSE_WRITE, ", 4096);
8.      if (event->mask & IN_CLOSE_NOWRITE)      strncat(buf,"CLOSE_NOWRITE, ", 4096);
9.      if (event->mask & IN_CREATE)         strncat(buf,"CREATE, ", 4096);
10.     if (event->mask & IN_DELETE)         strncat(buf,"DELETE, ", 4096);
11.     if (event->mask & IN_DELETE_SELF)    strncat(buf,"DELETE_SELF, ", 4096);
12.     if (event->mask & IN_MODIFY)         strncat(buf,"MODIFY, ", 4096);
13.     if (event->mask & IN_MOVE_SELF)      strncat(buf,"MOVE_SELF, ", 4096);
14.     if (event->mask & IN_MOVED_FROM)     strncat(buf,"MOVED_FROM, ", 4096);
```

inotify.c

```
1.      if (event->mask & IN_MOVED_TO)        strncat(buf,"MOVED_TO, ", 4096);
2.      if (event->mask & IN_OPEN)            strncat(buf,"OPEN, ", 4096);
3.      if (event->mask & IN_IGNORED)         strncat(buf,"IGNORED, ", 4096);
4.      if (event->mask & IN_ISDIR)           strncat(buf,"ISDIR, ", 4096);
5.      if (event->mask & IN_Q_OVERFLOW)      strncat(buf,"Q_OVERFLOW, ", 4096);
6.      buf[strlen(buf)-2]='\0';
7.      strncat(buf, "}", 4096);
8.      sprintf(buf, "%s cookie=%d", buf, event->cookie);
9.      if (event->len>0)
10.     sprintf(buf, "%s name = %s\n", buf, event->name);
11.     else
12.         sprintf(buf, "%s name = null\n", buf);
13.     printf("%s", buf);
14. }
```

結果

```
$ ./inotify ./list_acl.c ./list_acl_simple.c ..
[..] {OPEN, ISDIR} cookie=0 name = ch06
[..] {ACCESS, ISDIR} cookie=0 name = ch06
[..] {CLOSE_NOWRITE, ISDIR} cookie=0 name = ch06
```

小結

- inotify 是 Linux 特有的功能，其他 OS 雖然也有類似的功能，但必須自行查閱
  - 例如 Windows，使用這個 system call “FindFirstChangeNotification”，在 .NET 內可以使用 “ FileSystemWatcher”
- 這個功能用在目錄監看、自動處理等，相當好用
- Dropbox 就是用這套 API 實現即時監控的功能

作業

- 攔截一個目錄裡面的所有物件，必須使用遞迴，以包含子目錄
- 還需要攔截所有動作，並且顯示在螢幕上