Linux 作業系統

作業一

授課教授：許富皓 老師

組別: 第 23 組

成員: 資工碩一 104522010 施宇謙

資工碩一 104522040 張瑞慶

資工碩一 104522065 張翔琁

作業內容:

0.測試環境和版本

1.如何加入 system call

2.相關指令使用

3.Part I 程式和執行結果說明

4.Part II 程式和執行結果說明

檔案路徑說明:

Kernel system call : linux_survey_TT.c

報告: linux-作業 1-104522010-104522040-104522065_2

Part I : maps3660 (ps aux 印出 process 1 的資訊)

Part I : MATRIX3660.txt (test3.c 印出 process 1 的資訊)

Part II : maps3660 和 MATRIX3660.txt 同上

Part II : maps4854 (ps aux 印出 process 2 的資訊)

Part II : MATRIX4854.txt (test3.c 印出 process 2 的資訊)

Part II : same_p.txt (process 1 和 process 2 重疊的 physical address)

Part II : same_v.txt (process 1 和 process 2 重疊的 virtual address)

testbench : test3.c 是 Part I 和 Part II 的測試程式，t3 為其執行檔

testbench : test4.c 是 Part II 的測試程式，t4 為其執行檔

0.測試環境和版本

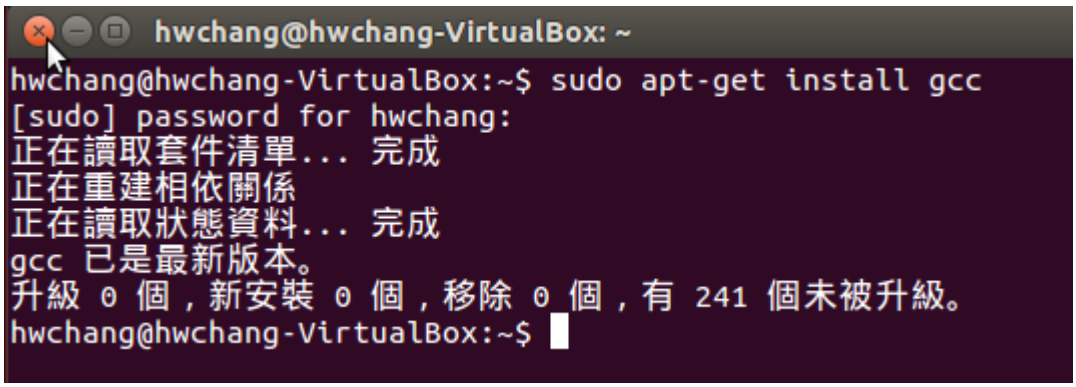
VM	**Virtual Box**
安裝 linux 發行版本	**Ubuntu 14.04(32-bit)**
編譯 Kernel 版本	**linux-3.10.77**

Ubuntu 12.10 的 apt-get 更新 list 無法下載 libncurses-dev，需額外新增 sudo gedit /etc/resolv.conf 底下的 nameserver 8.8.8.8 或新增 repository ，所以改用 Ubuntu 14.04。

1.如何加入 system call

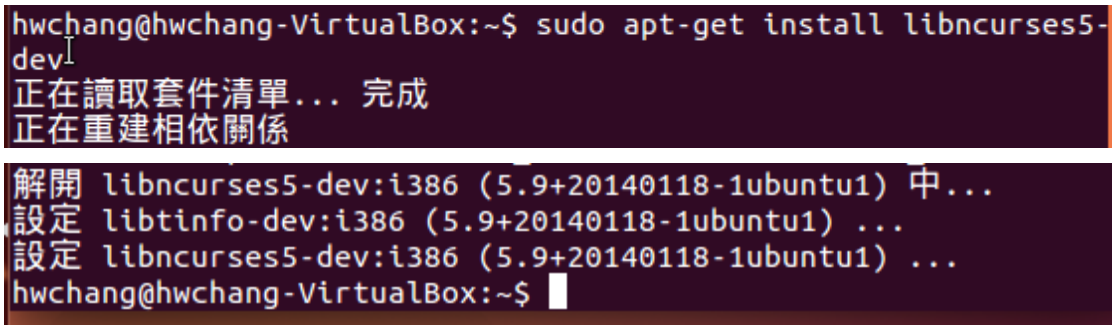
1.1 檢查有無安裝 gcc(一般來說已安裝)

sudo apt-get install gcc



1.2 安裝開發套件

sudo apt-get install libncurses5-dev



1.3 將軟體更新至最新版

sudo apt-get update && sudo apt-get upgrade

1.4 下載 kernel 檔案 linux-3.10.77.tar.gz

這邊我們直接用瀏覽器下載再解壓縮至/usr/src/

sudo tar zxvf linux-3.10.77.tar.gz -C /usr/src/

1.5 加入 system call

在/usr/src/linux-3.10.77/arch/x86/kernel 加入 linux_survey_TT.c 的程式，這個就是我們的 system call。

```
#include <linux/linkage.h>
#include <linux/kernel.h>
asmlinkage int sys_linux_survey_TT (int pid, char* data) {
        printk(KERN_EMERG "Hello world!");
        return 1;
}
```

編輯/usr/src/linux-3.10.77/include/linux/syscalls.h，到最後面增加

```
asmlinkage int sys_linux_survey_TT (int pid, char* data);
```

編輯/usr/src/linux-3.0.77/arch/x86/syscalls/syscall_32.tbl，到最後面加上

```
351     i386    linux_survey_TT     sys_linux_survey_TT
```

※編號為最後一號，依序遞增。

編輯/usr/src/linux-3.0.77/arch/x86/kernel/Makefile，加這一行：

```
obj-y                           += linux_survey_TT.o
```

※若要加速觀看自己的 system call 有無錯誤，可加至較前面的位置。

1.6 make kernel

以 root 身分切到資料夾

cd /usr/src/linux-3.10.77/

建立設定檔

make menuconfig

以 root 執行

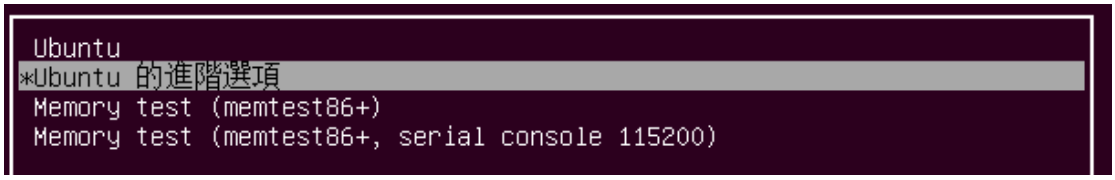
make && make modules_install && make install

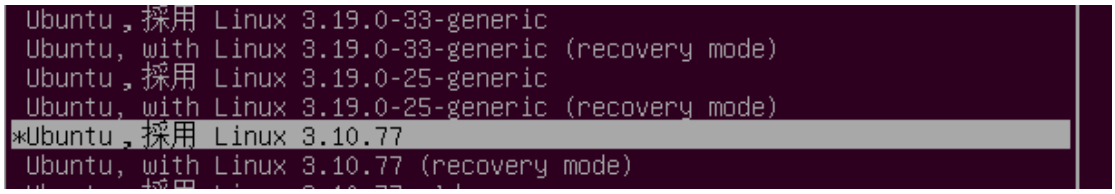
若有修改 system call 則清除設定檔和目的檔，並重新作設定

make mrproper && make menuconfig

按 shift 進開機選單

選擇 ubuntu 進階選項和對應 kernel 版本





加入小型測試程式

```c
#include <stdio.h>
#include <unistd.h>
#define REPEAT_TIME 10
#define MEMORY_SIZE 10000

#define linux_survey_TT(pid, result) syscall(351, pid, result)
main(){
        int i, pid;
        char result[MEMORY_SIZE];
        printf("Input the PID of the process that you want to observe:");
        scanf("%d", &pid);
        linux_survey_TT(pid, result);
}
~
```

執行測試程式並呼叫 system call:

syscall(system call 號碼, system call 參數…)

※若執行失敗會回傳-1。

※若非法存取記憶體，執行程式會回傳 killed(已砍掉)。

執行結果(利用 dmesg –C 先把 debug message 清除，再跑測試程式):

```
hwchang@hwchang-VirtualBox:~/文件$ sudo dmesg -C
hwchang@hwchang-VirtualBox:~/文件$ ./hello_world
Input the PID of the process that you want to observe:2000
hwchang@hwchang-VirtualBox:~/文件$ dmesg
[ 2631.697309] Hello world!
hwchang@hwchang-VirtualBox:~/文件$
```

2.相關指令使用

2.1 開啟個別 process 的 firefox(每個 firefox 為一個獨立的 process)

每次開啟 firefox 會讀取預設的 profile，欲將每個 firefox 為獨立的 process，產生時需讀取不同的 profile。

建立 profile

firefox –p

以不同的 profile 開啟 firefox(建議放至背景執行，終端機可繼續下指令)

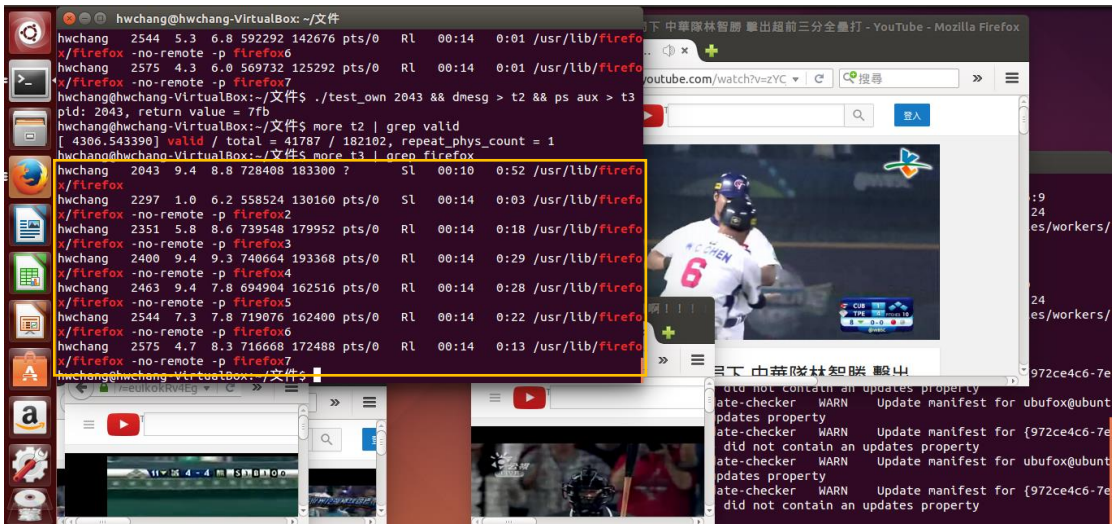
firefox –no-remote –P firefox2 &

新產生的 profile 會放在以下路徑，可檢查有無新增成功:

/home/使用者帳號名稱/.mozilla/firefox/xxxx.firefox2　firefox2 是新的設定檔名稱

或是利用 firefox –p 再次確認

執行結果:



2.2 查看 kernel 版本

uname –a

```
hwchang@hwchang-VirtualBox:~/文件$ uname -a
Linux hwchang-VirtualBox 3.10.77 #4 SMP Thu Nov 19 17:53:02 CST 2015 i686 i686 i
686 GNU/Linux
```

2.3 查看 process 的 pid

使用 ps aux | grep 應用程式名稱

或者使用 pidof 應用程式名稱

如 ps aux | grep firefox 和 pidof firefox

```
hwchang@hwchang-VirtualBox:~$ pidof firefox
23960
```

2.4 觀看 process 的 virtual address space

cat /proc/PID/maps

```
b777d000-b777e000 r--p 0001d000 08:01 1185130    /usr/lib/firef
ox/firefox
b777e000-b777f000 rw-p 0001e000 08:01 1185130    /usr/lib/firef
ox/firefox
bfe18000-bfe3a000 rw-p 00000000 00:00 0          [stack]
hwchang@hwchang-VirtualBox:~$
```

2.5 觀看機器的總記憶體資訊

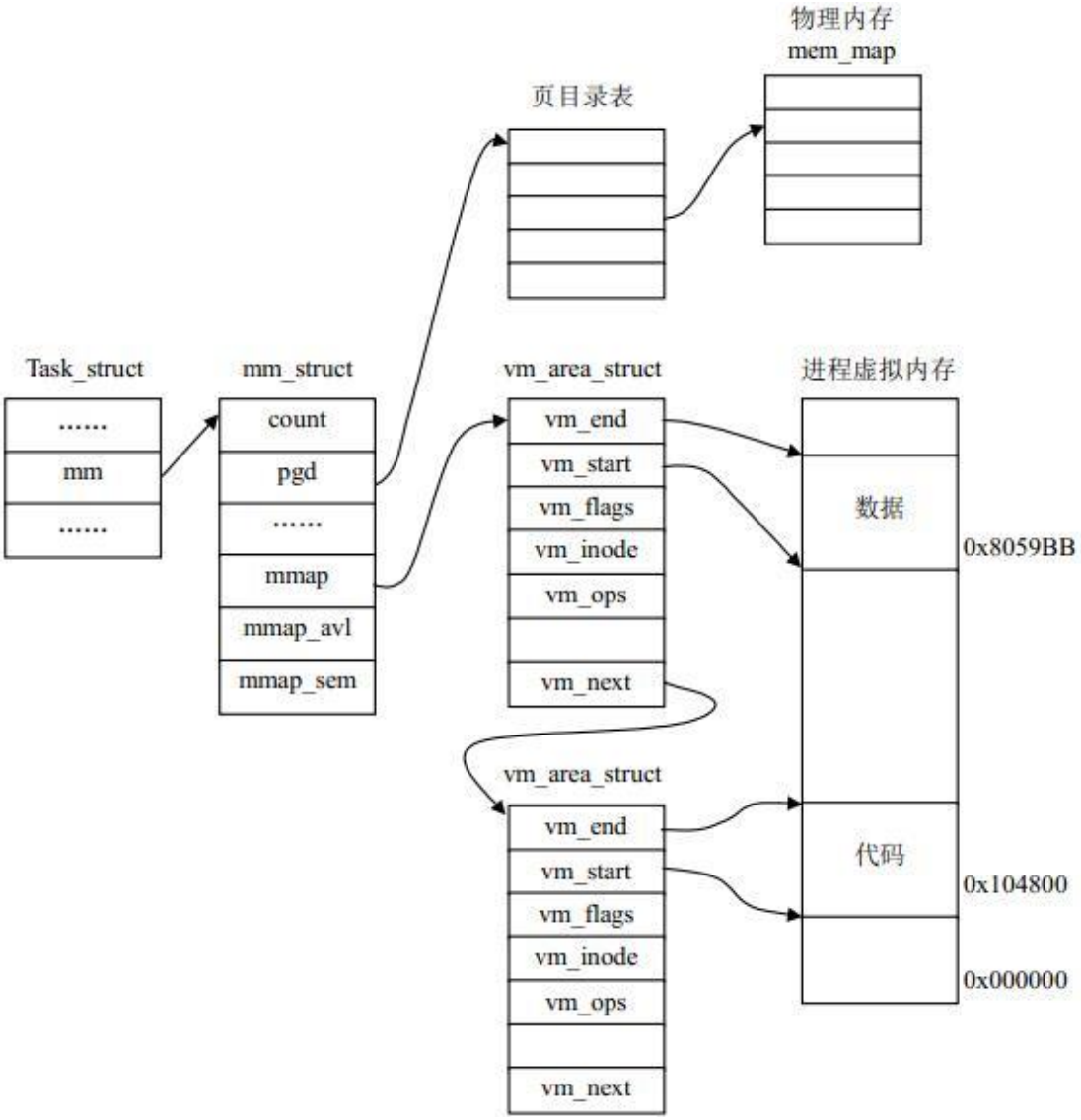
cat /proc/meminfo

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# cat /proc/meminfo
MemTotal:        3109004 kB
MemFree:         1924184 kB
Buffers:           66468 kB
Cached:           509356 kB
SwapCached:            0 kB
Active:           659056 kB
Inactive:         451132 kB
Active(anon):     535076 kB
Inactive(anon):     9968 kB
Active(file):     123980 kB
Inactive(file):   441164 kB
Unevictable:          32 kB
Mlocked:              32 kB
HighTotal:       2236360 kB
HighFree:        1175620 kB
LowTotal:         872644 kB
LowFree:          748564 kB
SwapTotal:       3142652 kB
SwapFree:        3142652 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        532488 kB
Mapped:           131348 kB
Shmem:             10684 kB
Slab:              44248 kB
SReclaimable:      27348 kB
SUnreclaim:        16900 kB
KernelStack:        3544 kB
PageTables:         8208 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     4697152 kB
Committed_AS:    3145780 kB
VmallocTotal:     122880 kB
VmallocUsed:        4412 kB
VmallocChunk:     116448 kB
AnonHugePages:    217088 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:        8184 kB
DirectMap2M:      901120 kB
```

3.Part I 程式和執行結果說明

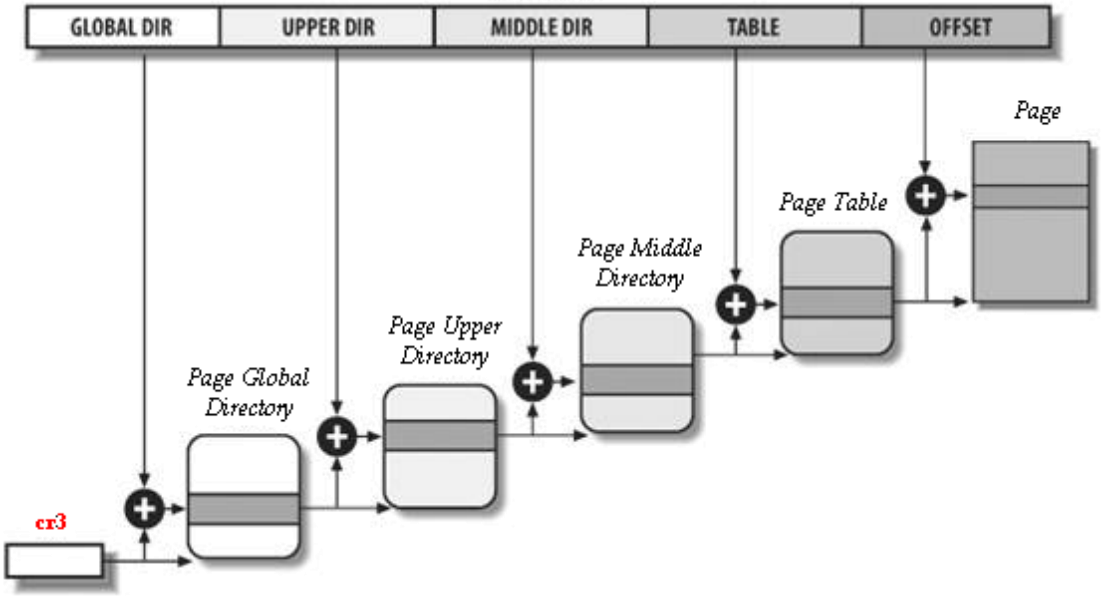
3.1 做法介紹

首先透過find_task_by_vpid(pid)，透過輸入pid得到該pid的struct task_struct。再透過 task 的 struct mm_struct 欄位取得記憶體的分布情形，再透過 struct mm_struct 裡面的 struct vm_area_struct 欄位去找尋虛擬記憶體分配，最後透過每一個 vm_area_struct 的 start 與 end 欄位得到虛擬記憶體分配的區塊，再用 vm_next 欄位尋找其他串連在一起的 struct vm_area_struct 如下圖，可以找出全部的虛擬記憶體分配。



接下來透過虛擬記憶體分配的區間資訊與 mm_struct 可以繼續往下求得實體位置，透過 mm_struct 的 pgd 欄位可以求得 pgd 基底位置，接下來透過 pgd_offset(mm,vaddr)，可得透過 pgd 基底位置與 vaddr 的 pgd offset 算出是哪個欄位，然後回傳該欄位內容。然後再接著使用 pud_offset(pgd,vaddr)、pmd_offset(pud,vaddr)、pte_offset_map(pmd,vaddr)，可以尋找到 pte 位置，最後透過 pte_page(pte)去找到描述該 page 的 struct pages。

然後就可以從 page_to_phys(page)去找到實體記憶體位置。如果再分配的時候 PSE = 1 分配頁面就會變成大頁，所以需要在加上!pmd_large(*pmd)，當這個敘述成立的時候代表他是小頁，可以進入 pte_offset_map()，如果不成立代表他是大頁。



最後透過字元陣列回傳虛擬區間跟實體區間還有虛擬量與實體量如下圖

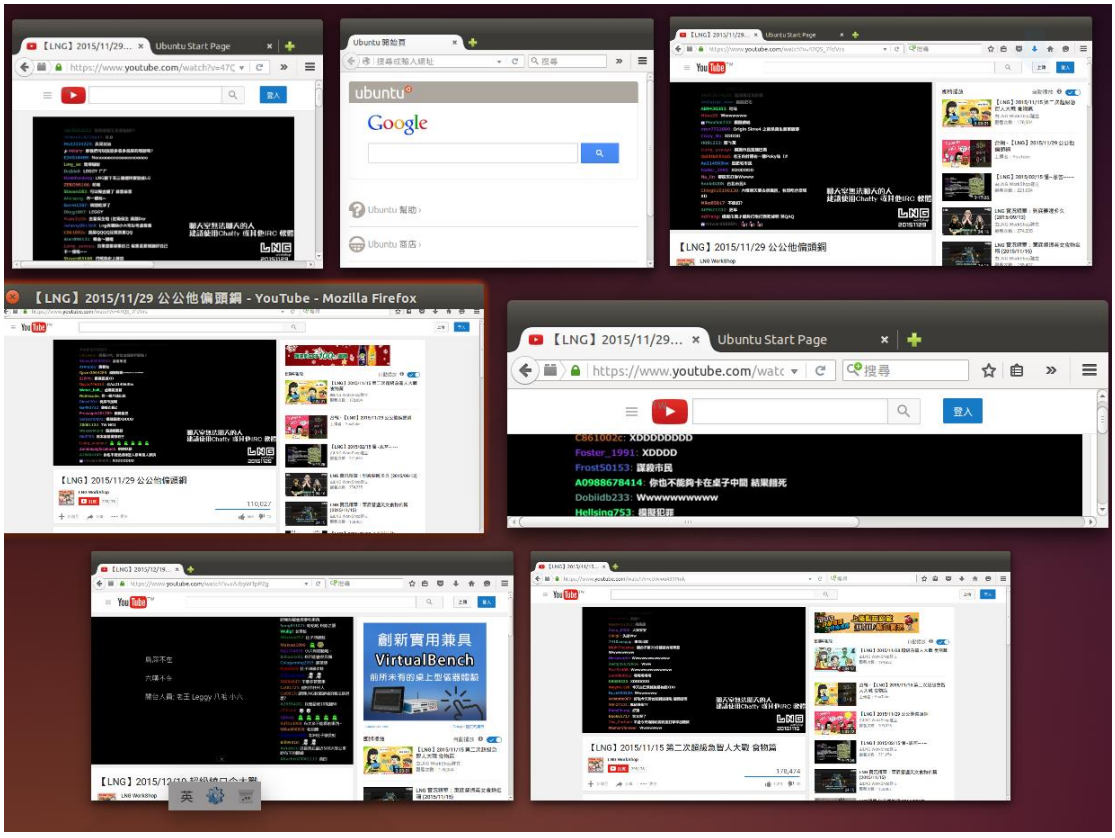
```
void write_addr(unsigned long int addr,char *buffer)
{

    int i = 0;
    if(write_count < 2500)
    {
    for(i = 0; i < 4; i++)
    {
        buffer[write_count*4+i] = addr >> (3-i) * 8;
    }
    write_count++;
    }
}
write_addr(i,output);
write_addr(p_mem_count,output);
write_addr(for_linux_pmd_count,output);
write_addr(0xffffffff,output);
total = 696272 paddrt = 103604 pmaddrt = 34820 paddr = 138424
```

3.2 **partI**

開啟第一個 firefox 在圖中為 jerrych+開啟的 PID 為 3660，另外六個為 root 開啟的 firefox –P 的六個正在觀看 youtube 的 firefox

```
jerrych+  3660  2.9  4.0 667804 124800 ?       Sl   18:52   0:03 /usr/lib/firefox/firefox
root      3743 14.0  5.2 738368 163320 pts/27  Sl+  18:52   0:14 /usr/lib/firefox/firefox -P
root      3866 16.8  3.3 494196 104836 pts/27  Sl+  18:52   0:16 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 3743 tr
root      3898  7.6  4.5 703188 142500 pts/24  Sl+  18:53   0:05 /usr/lib/firefox/firefox -P
root      4022 12.9  2.6 450692 80876 pts/24   Sl+  18:53   0:09 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 3898 tr
root      4051 10.6  4.8 711828 150700 pts/26  Sl+  18:53   0:06 /usr/lib/firefox/firefox -P
root      4181 14.6  3.2 493528 99700 pts/26   Sl+  18:53   0:08 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4051 tr
root      4213 14.4  4.9 713656 154828 pts/25  Sl+  18:53   0:07 /usr/lib/firefox/firefox -P
root      4338 22.3  3.2 491340 102524 pts/25  Sl+  18:53   0:10 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4213 tr
root      4370 17.8  4.9 764652 154712 pts/11  Sl+  18:53   0:06 /usr/lib/firefox/firefox -P
root      4502 16.2  3.1 493528 99376 pts/11   Sl+  18:53   0:04 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4370 tr
root      4532 31.3  4.7 740312 147316 pts/0   Sl+  18:54   0:05 /usr/lib/firefox/firefox -P
root      4651 21.5  3.1 487384 97756 pts/0    Sl+  18:54   0:02 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4532 tr
root      4689  0.0  0.0   4696   824 pts/5    S+   18:54   0:00 grep --color=auto firefox
root@jerrychang-VirtualBox:/home/jerrychang/testLinux#
```



透過 test bench 測試該 system call 是否正常運作，撈出 process1(PID = 3660)的 Virtual 區間與對應的實體區間以及實體頁面在虛擬頁面所占的比例數。

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# ./t3 3660 MATRIX3660.txt && ps aux | grep 3660
3660
total = 160836 paddrt = 20489 pmaddrt = 14848
total = 643344 paddrt = 81956 paddrt = 141348
jerrych+  3660  0.9  4.4 643344 138492 ?       Sl   18:52   0:04 /usr/lib/firefox/firefox
root      4746  0.0  0.0   4696   828 pts/5    S+   18:59   0:00 grep --color=auto 3660
root@jerrychang-VirtualBox:/home/jerrychang/testLinux#
```

從 t3 test3.c 這個 test bench 可以看出 total virtual address 量是 643344，physical 是 141348。所以百分比為 141348/643344 約等於 21.9%的虛擬記憶體有實體記憶體，從 ps aux 指令可以看出虛擬記憶體為相同數量，實體則多了一點。而列出的虛擬記憶體與實體記憶體分配則寫成 MATRIX3743.txt 這個檔案，虛擬的區間是依照 cat /proc/3660/maps > maps3660 所列出的區間去列出，下圖為 MATRIX3660.txt 與 maps3660 的部分截圖。

```
vaddr = 8d500000-8d500fff  paddr = 0-0
vaddr = 8d501000-8d501fff  paddr = 0-0
vaddr = 8d502000-8d502fff  paddr = 0-0
vaddr = 8d503000-8d503fff  paddr = 0-0
vaddr = 8d504000-8d504fff  paddr = 0-0
vaddr = 8d505000-8d505fff  paddr = 0-0
vaddr = 8d506000-8d506fff  paddr = 0-0
vaddr = 8d507000-8d507fff  paddr = 0-0
vaddr = 8d508000-8d508fff  paddr = 0-0
vaddr = 8d509000-8d509fff  paddr = 0-0
```

```
8d4ff000-8d500000 ---p 00000000 00:00 0
8d500000-8df00000 rw-p 00000000 00:00 0          [stack:3742]
8f000000-8f400000 rw-p 00000000 00:00 0
8f4fe000-8f4ff000 ---p 00000000 00:00 0
8f4ff000-8fcff000 rw-p 00000000 00:00 0
8fcff000-8fd00000 ---p 00000000 00:00 0
8fd00000-90800000 rw-p 00000000 00:00 0
908ad000-90cff000 r--p 00000000 08:01 4195976    /usr/share/fonts/truetype/droid/DroidSansFallbackFull.ttf
90cff000-90d00000 ---p 00000000 00:00 0
90d00000-91600000 rw-p 00000000 00:00 0          [stack:3735]
```

4.Part II 程式和執行結果說明

再新增一個 firefox 為 process 2 並且也跑一個 youtube 影片，透過比較兩個 ps aux | grep firefox 結果找出 process2 的 PID 為 5086。

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# ps aux | grep firefox
jerrych+  3660  0.3  4.6 642320 143780 ?       Sl   18:52   0:05 /usr/lib/firefox/firefox
root      3743  6.2  5.7 740932 178628 pts/27  Sl+  18:52   1:25 /usr/lib/firefox/firefox -P
root      3866 15.0  3.6 500340 112092 pts/27  Sl+  18:52   3:25 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 3743 tr
root      3898  2.2  5.3 708960 164796 pts/24  Sl+  18:53   0:30 /usr/lib/firefox/firefox -P
root      4022 13.8  2.8 454788 88204 pts/24   Sl+  18:53   3:05 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 3898 tr
root      4051  2.7  5.5 717344 171524 pts/26  Sl+  18:53   0:36 /usr/lib/firefox/firefox -P
root      4181 14.5  3.4 501720 105976 pts/26  Sl+  18:53   3:14 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4051 tr
root      4213  2.9  5.0 719364 155948 pts/25  Sl+  18:53   0:39 /usr/lib/firefox/firefox -P
root      4338 15.5  3.7 507724 116524 pts/25  Rl+  18:53   3:24 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4213 tr
root      4370  2.7  4.7 696660 146484 pts/11  Sl+  18:53   0:36 /usr/lib/firefox/firefox -P
root      4502 14.5  3.3 499672 105012 pts/11  Sl+  18:53   3:09 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4370 tr
root      4532  2.8  4.6 713236 144464 pts/0   Sl+  18:54   0:36 /usr/lib/firefox/firefox -P
root      4651 14.7  3.4 497624 105952 pts/0   Sl+  18:54   3:10 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4532 tr
root      4854 25.4  5.0 759076 156544 pts/28  Sl+  19:14   0:08 /usr/lib/firefox/firefox -P
root      4982 27.6  3.0 491144 93912 pts/28   Sl+  19:15   0:05 /usr/lib/firefox/plugin-cont
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4854 tr
root      5016  0.0  0.0   4696   828 pts/5    S+   19:15   0:00 grep --color=auto firefox
root@jerrychang-VirtualBox:/home/jerrychang/testLinux#
```

將 process2 的虛擬與實體記憶體一併記錄下來，可以從結果看出跟 ps aux 比較結果，虛擬記憶體量相同，實體記憶體稍微多一點點，再透過 cat /proc/4854/maps > maps4854，與 test bench(t3)所產生的 MATRIX4854.txt 做比較，而虛擬記憶體被轉成實體的比率為 21.8%如下圖。

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# ./t3 4854 MATRIX4854.txt && ps aux | grep 4854
4854
total = 180951 paddrt = 36375 pmaddrt = 3072
total = 723804 paddrt = 145500 pmaddrt = 12288 paddr = 157788
root      4854  4.1  4.9 723804 155420 pts/28  Sl+  19:14   0:12 /usr/lib/firefox/firefox -P
root      4982 16.1  3.3 501384 103132 pts/28  Sl+  19:15   0:44 /usr/lib/firefox/plugin-container /us
ox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir /usr/lib/firefox/browser 4854 true plugin
root      5077  0.0  0.0   4696   824 pts/5    R+   19:19   0:00 grep --color=auto 4854
root@jerrychang-VirtualBox:/home/jerrychang/testLinux#
```

```
vaddr = 88100000-88100fff  paddr = 0-0
vaddr = 88101000-88101fff  paddr = 0-0
vaddr = 88102000-88102fff  paddr = 0-0
vaddr = 88103000-88103fff  paddr = 0-0
vaddr = 88104000-88104fff  paddr = 0-0
vaddr = 88105000-88105fff  paddr = 0-0
vaddr = 88106000-88106fff  paddr = 0-0
vaddr = 88107000-88107fff  paddr = 0-0
vaddr = 88108000-88108fff  paddr = 0-0
vaddr = 88109000-88109fff  paddr = 0-0
```

```
88100000-88200000 rw-p 00000000 00:00 0
882ff000-88300000 ---p 00000000 00:00 0
88300000-88e00000 rw-p 00000000 00:00 0          [stack:5002]
89efd000-89efe000 ---p 00000000 00:00 0
89efe000-8a6fe000 rw-p 00000000 00:00 0
8a6fe000-8a6ff000 ---p 00000000 00:00 0
8a6ff000-8aeff000 rw-p 00000000 00:00 0          [stack:4998]
8aeff000-8af00000 ---p 00000000 00:00 0
8af00000-8c100000 rw-p 00000000 00:00 0          [stack:4997]
8c1ff000-8c200000 ---p 00000000 00:00 0
```

接著再跑一次 process1 的結果，從結果可以看出虛擬、實體記憶體多了一點點，比例下降為 21.5%，比例下降如下圖，記憶體區間分配如下圖。

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# ./t3 3660 MATRIX3660.txt && ps aux | grep 3660
3660
total = 164948 paddrt = 28385 pmaddrt = 7168
total = 659792 paddrt = 113540 pmaddrt = 28672 paddr = 142212
jerrych+  3660  0.3  4.4 659792 139464 ?       Sl   18:52   0:07 /usr/lib/firefox/firefox
root      5133  0.0  0.0   4696   824 pts/5    R+   19:25   0:00 grep --color=auto 3660
root@jerrychang-VirtualBox:/home/jerrychang/testLinux#
```

```
vaddr = 8d500000-8d500fff  paddr = 0-0
vaddr = 8d501000-8d501fff  paddr = 0-0
vaddr = 8d502000-8d502fff  paddr = 0-0
vaddr = 8d503000-8d503fff  paddr = 0-0
vaddr = 8d504000-8d504fff  paddr = 0-0
vaddr = 8d505000-8d505fff  paddr = 0-0
vaddr = 8d506000-8d506fff  paddr = 0-0
vaddr = 8d507000-8d507fff  paddr = 0-0
vaddr = 8d508000-8d508fff  paddr = 0-0
vaddr = 8d509000-8d509fff  paddr = 0-0
```

```
8d500000-8df00000 rw-p 00000000 00:00 0          [stack:3742]
8f000000-8f400000 rw-p 00000000 00:00 0
8f4fe000-8f4ff000 ---p 00000000 00:00 0
8f4ff000-8fcff000 rw-p 00000000 00:00 0
8fcff000-8fd00000 ---p 00000000 00:00 0
8fd00000-90800000 rw-p 00000000 00:00 0
908ad000-90cff000 r--p 00000000 08:01 4195976    /usr/share/fonts/truetype/droid/DroidSansFallbackFull.ttf
90cff000-90d00000 ---p 00000000 00:00 0
90d00000-91600000 rw-p 00000000 00:00 0          [stack:3735]
916fe000-916ff000 ---p 00000000 00:00 0
```

透過 testbench2(test4.c t4)跑出實體位置與虛擬位置，分別為兩檔案 same_v.txt 與 same_p.txt 分別為相同的 virtual 位置與相同的 physical 位置，其中編號 1 為起始位置，編號 2 為結束位置，結果如下圖。

```
root@jerrychang-VirtualBox:/home/jerrychang/testLinux# ./t4 MATRIX3660.txt MATRIX4854.txt
i = 165513
```

```
vaddr1 : 8d4ff000 paddr1 : a5bff000
vaddr2 : 8d4fffff paddr2 : a5bfffff
vaddr1 : 8d500000 paddr1 : a4005000
vaddr2 : 8d500fff paddr2 : a4005fff
vaddr1 : 8d501000 paddr1 : a4033000
vaddr2 : 8d501fff paddr2 : a4033fff
vaddr1 : 8d502000 paddr1 : a4034000
vaddr2 : 8d502fff paddr2 : a4034fff
vaddr1 : 8d503000 paddr1 : a4035000
vaddr2 : 8d503fff paddr2 : a4035fff
```

我們透過 process1 與 process2 的 maps 去看相同的 Share Library

```
96da6000-96ef9000 r-xp 00000000 08:01 3282734    /usr/lib/i386-linux-gnu/libxml2.so.2.9.1
95ea5000-95ff8000 r-xp 00000000 08:01 3282734    /usr/lib/i386-linux-gnu/libxml2.so.2.9.1
```

96da6000 為 process1 的虛擬記憶體位置，透過 MATRIX3660.txt 找到轉出來的實體記憶體，95ea5000 為 process2 的虛擬記憶體位置，透過 MATRIX4854.txt 找到轉出來的實體記憶體兩個皆為 a0171000，再用 a0171000 去搜尋 same_p，確實存在驗證成功。

```
vaddr = 96da6000-96da6fff  paddr = a0171000-a0171fff

vaddr = 95ea5000-95ea5fff  paddr = a0171000-a0171fff  paddr2 : a0cb9fff
                                                      paddr1 : a0171000
                                                      paddr2 : a0171fff
```

參考資料:
[1] firefox 設定檔方法
http://wiki.moztw.org/%E8%A8%AD%E5%AE%9A%E6%AA%94%E8%A9%B3%E8%A7%A3#ProfileManager:Linux
[2]how to fix “apt-get update” failed to fetch error?
http://askubuntu.com/questions/298177/a-failed-to-fetch-error-occurs-when-apt-get-update-is-run-how-do-i-fix-this
[3]how to get to the GRUB menu at boot-time?
http://askubuntu.com/questions/16042/how-to-get-to-the-grub-menu-at-boot-time
[4]vm_area_struct 管理 virtual address 方法圖
http://whitepig-wordpress.stor.sinaapp.com/uploads/2013/09/20130901230716.jpg
[5]許富皓老師投影片-Memory Addressing
http://www.csie.ncu.edu.tw/~hsufh/COURSES/FALL2015/linuxLecture_3_9-4.ppt

附上 system call 程式 linux_survey_TT.c:

```
#include <linux/linkage.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <asm/pgtable.h>
#include <asm/pgtable_types.h>
#include <linux/highmem.h>

unsigned long int for_linux_pmd_count = 0, for_linux_pte_NULL = 0, write_count = 0;
unsigned char wflag = 0;

void write_addr(unsigned long int addr,char *buffer)
{

	int i = 0;
	if(write_count < 500000)
	{
	for(i = 0; i < 4; i++)
	{
		buffer[write_count*4+i] = addr >> (3-i) * 8;
	}
	write_count++;
	}
}
```

```
static struct page *
my_follow_page(struct mm_struct *mm, unsigned long addr, char *buffer, unsigned long int *ppaddr)
{
pud_t *pud;
pmd_t *pmd;
pgd_t *pgd;
pte_t pte, *ptep;
long unsigned    int paddr = 0;
struct page *page = NULL;
phys_addr_t phy = 0;
pgd = pgd_offset(mm, addr);
if (pgd_present(*pgd)) {
        pud = pud_offset(pgd,addr);
        if (pud_present(*pud)) {
                pmd = pmd_offset(pud,addr);
                if(pmd_present(*pmd))
                {
                        if(!pmd_large(*pmd))
                        {
                                ptep = pte_offset_map(pmd,addr);
                                if (pte_none(*ptep) || ptep == NULL)
                                {
                                        printk("pte_NULL\n");
                                        for_linux_pte_NULL++;
                                }
                                else
                                {
                                        pte = *ptep;
                                        if(pte_present(*ptep))
                                        {
                                             paddr = (pte_val(*ptep) & PAGE_MASK) | (addr & ~PAGE_MASK);
                                             printk(KERN_EMERG"physical address of 0x%lx is 0x%lx
\n",addr,paddr);
                                             printk("__pa(0x%lx) = 0x%lx",addr,__pa(addr));
                                             page = pte_page(pte);
                                             phy = page_to_phys(page);
                                             paddr = (long unsigned int)phy;
                                             printk(KERN_EMERG "page frame is %lx",(unsigned long int)pte.pte);
```

```
                        //write_addr((long unsigned int)phy,buffer);
                        /*if(wflag == 0)
                        {
                            write_addr((long unsigned int)phy,buffer);
                            wflag = 1;
                        }*/
                    }
                }
                pte_unmap(ptep);
            }
            else
            {
                paddr = (pmd_val(*pmd) & PMD_MASK) | (addr & ~PMD_MASK);
                printk("Use Large Page PSE = 1\n");
                printk("physical address of 0x%lx is 0x%lx\n", addr, paddr);
                printk("__pa(vaddr) is 0x%lx", __pa(addr));
                //phy = paddr;
                /*if(wflag == 0)
                {
                    write_addr((long unsigned int)paddr,addr,buffer);
                    wflag = 1;
                }*/
                //write_addr((long unsigned int)paddr,buffer);
                //prev_paddr = paddr;
                for_linux_pmd_count++;

            }
        }
    }
}
//write_addr((long unsigned int)addr,buffer);//start vaddr
//write_addr((long unsigned int)paddr,buffer);//physical
*ppaddr = paddr;
return page;


}
```

```
asmlinkage void linux_survey_TT(int input,char *output) {
	struct task_struct *pid_task;
	struct mm_struct *pid_mm;
	struct vm_area_struct *pid_mmap;
	struct page *pid_page = NULL;
	struct page *last_page = NULL;
	phys_addr_t phy,last_phy = 0;
	unsigned long int i = 0,p_mem_count = 0,p_mem_include_0 = 0,paddr = 0;
	unsigned long int addr,prevaddr = 0;
	unsigned long int count = 0;
	/*pgd_t *pgd;
	pmd_t *pmd;
	pte_t *pte;
	pud_t *pud;*/

	pid_task = find_task_by_vpid(input);
	pid_mm     = pid_task->mm;
	pid_mmap = pid_mm->mmap;
	printk(KERN_EMERG "Hello World2 pid = %d \n",input);
	printk(KERN_EMERG "hello world2_new task %d\npid = %d\n",input,pid_task->pid);

	while(1)
	{
			addr = pid_mmap->vm_start;
			count = 0;
			wflag = 0;
			phy = 0;
			paddr = 0;
			//write_addr((long unsigned int)addr,output);//start vaddr
			while(1)
			{
				printk(KERN_EMERG "H2 LINE : %ld vm_start = %lx vm_end = %lx
\n",i,addr,addr + 4096);

				pid_page = my_follow_page(pid_mm,addr,output,&paddr);
				write_addr((long unsigned int)addr,output);
				write_addr((long unsigned int)paddr,output);
```

```
if(pid_page == NULL){
        printk(KERN_EMERG "pid_PAGE = NULL\n");
        /*if(wflag == 0){
                write_addr((long unsigned int)phy,output);//start paddr
                wflag = 1;
        }*/

}
else
{

        phy = page_to_phys(pid_page);
        /* if(wflag == 0){
                write_addr((long unsigned int)phy,output);//start paddr
                wflag = 1;
        }*/
        printk(KERN_EMERG "address = %lx\n",(unsigned long int)phy);
        if(phy != 0)
        {
                if(last_phy != phy)
                {
                        p_mem_count++;
                        last_phy = phy;
                }
        }
        if(pid_page != last_page)
        p_mem_include_0++;
        last_page = pid_page;
}
count++;
//if(prevaddr != addr)
i++;
prevaddr++;
// prevaddr = addr;
if(pid_mmap->vm_end - addr == 4096){
     i--;
}
if(pid_mmap->vm_end - addr >= 4096){
```

```
                        addr = pid_mmap->vm_start + 4096 * count;phy = 0;
                    }
                    else
                        break;
                }
               // write_addr((long unsigned int)addr,output);//end vaddr
               // write_addr((long unsigned int)phy,output);// end paddr

                if(pid_mmap->vm_next == NULL)
                    break;
                else
                    pid_mmap = pid_mmap->vm_next;

        }

        printk(KERN_EMERG "total = %ld py = %ld py_include0 = %ld pmd_count = %ld pte_NULL
= %ld prevaddr = %ld
\n",i,p_mem_count,p_mem_include_0,for_linux_pmd_count,for_linux_pte_NULL,prevaddr);
        write_addr(i,output);
        write_addr(p_mem_count,output);
        write_addr(for_linux_pmd_count,output);
        write_addr(0xffffffff,output);
        for_linux_pmd_count = 0;
        for_linux_pte_NULL    = 0;
        write_count = 0;

}
```