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 張翔珳

**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**

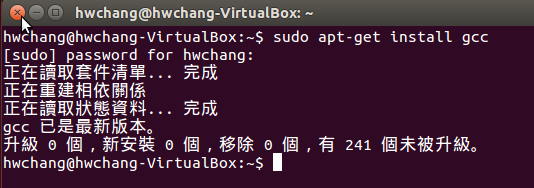
|  |  |
| --- | --- |
| **VM** | **Virtual Box** |
| **安裝linux發行版本** | **Ubuntu 14.04(32-bit)** |
| **編譯Kernel版本** | **linux-3.10.77** |

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**

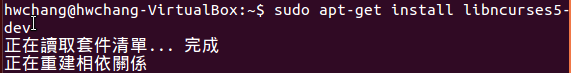
**1.1檢查有無安裝gcc(一般來說已安裝)**

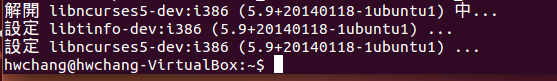
**sudo apt-get install gcc**



**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

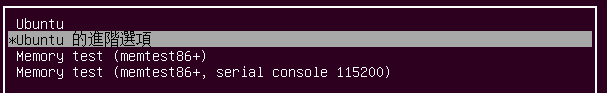
make && make modules\_install && make install

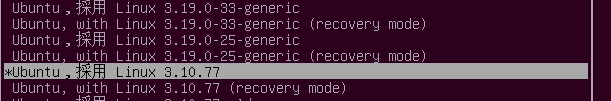
若有修改system call則清除設定檔和目的檔，並重新作設定

make mrproper && make menuconfig

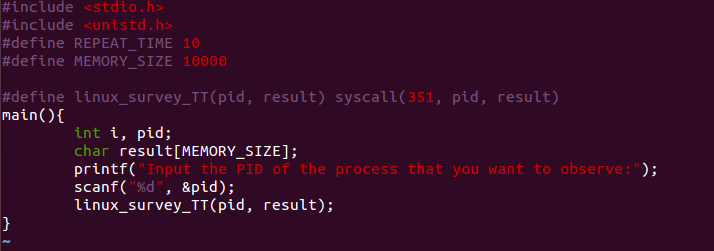
按shift進開機選單

選擇ubuntu進階選項和對應kernel版本





**加入小型測試程式**



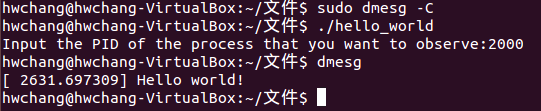
執行測試程式並呼叫system call:

**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):**



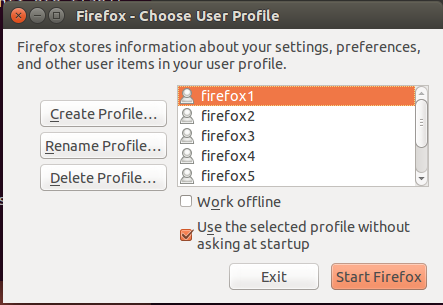
**2.相關指令使用**

**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)

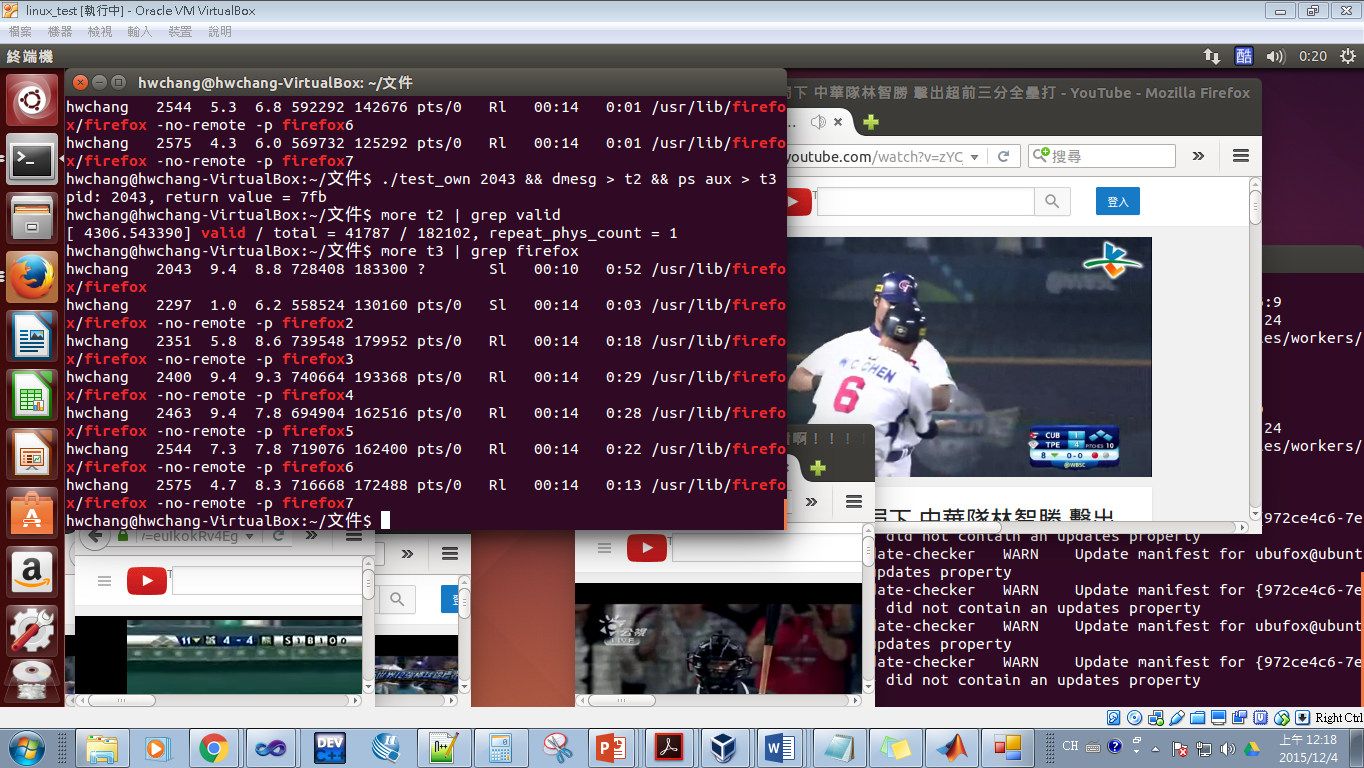
firefox –no-remote –P firefox2 &

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



**2.3查看process的pid**

使用ps aux | grep 應用程式名稱

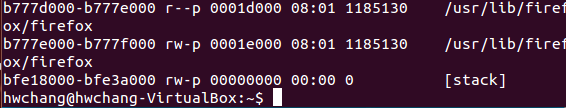
或者使用pidof應用程式名稱

如ps aux | grep firefox和pidof firefox



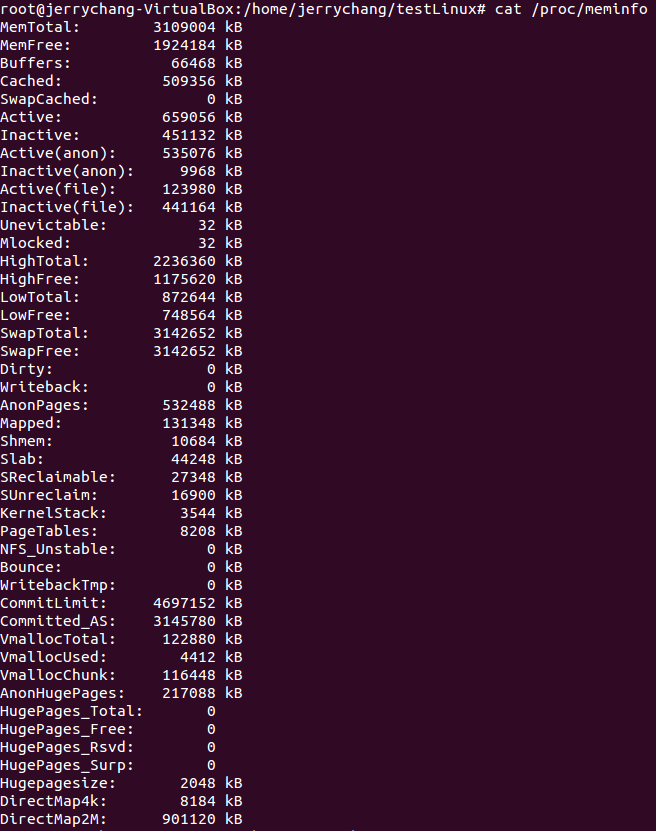
**2.4觀看process的virtual address space**

cat /proc/PID/maps



**2.5觀看機器的總記憶體資訊**

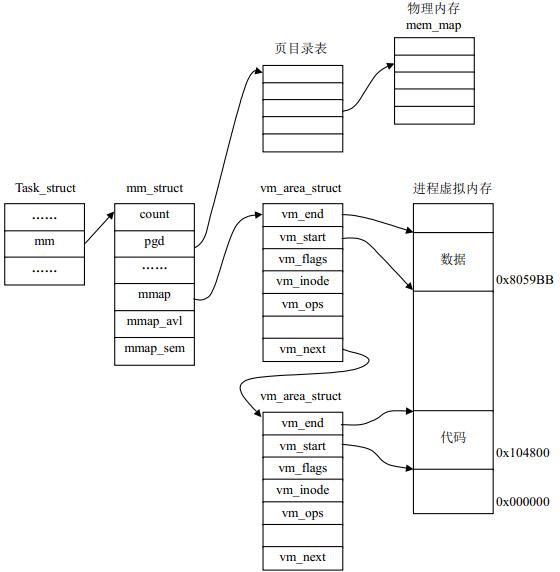
cat /proc/meminfo



**3.Part I程式和執行結果說明**

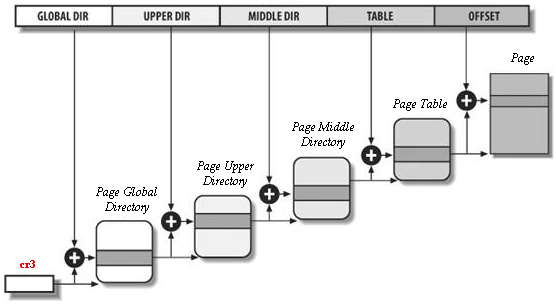
**3.1 做法介紹**

**首先透過**[find\_task\_by\_vpid](http://lxr.free-electrons.com/ident?v=3.10;i=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。

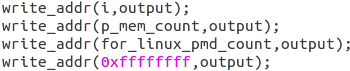
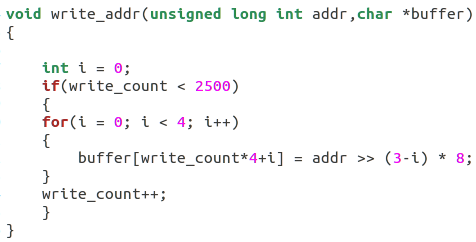
****

**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。

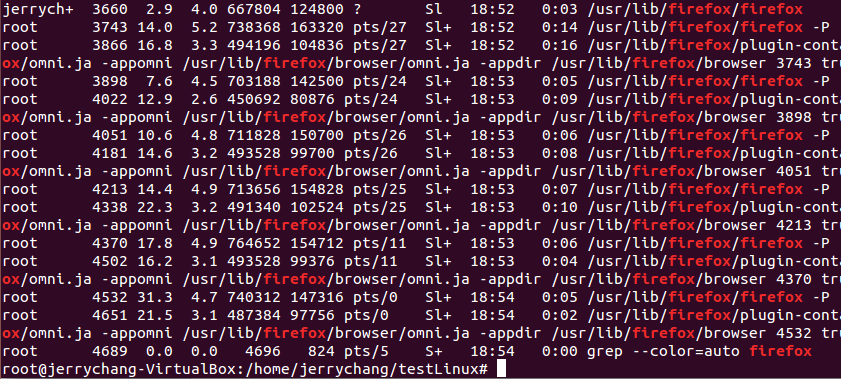


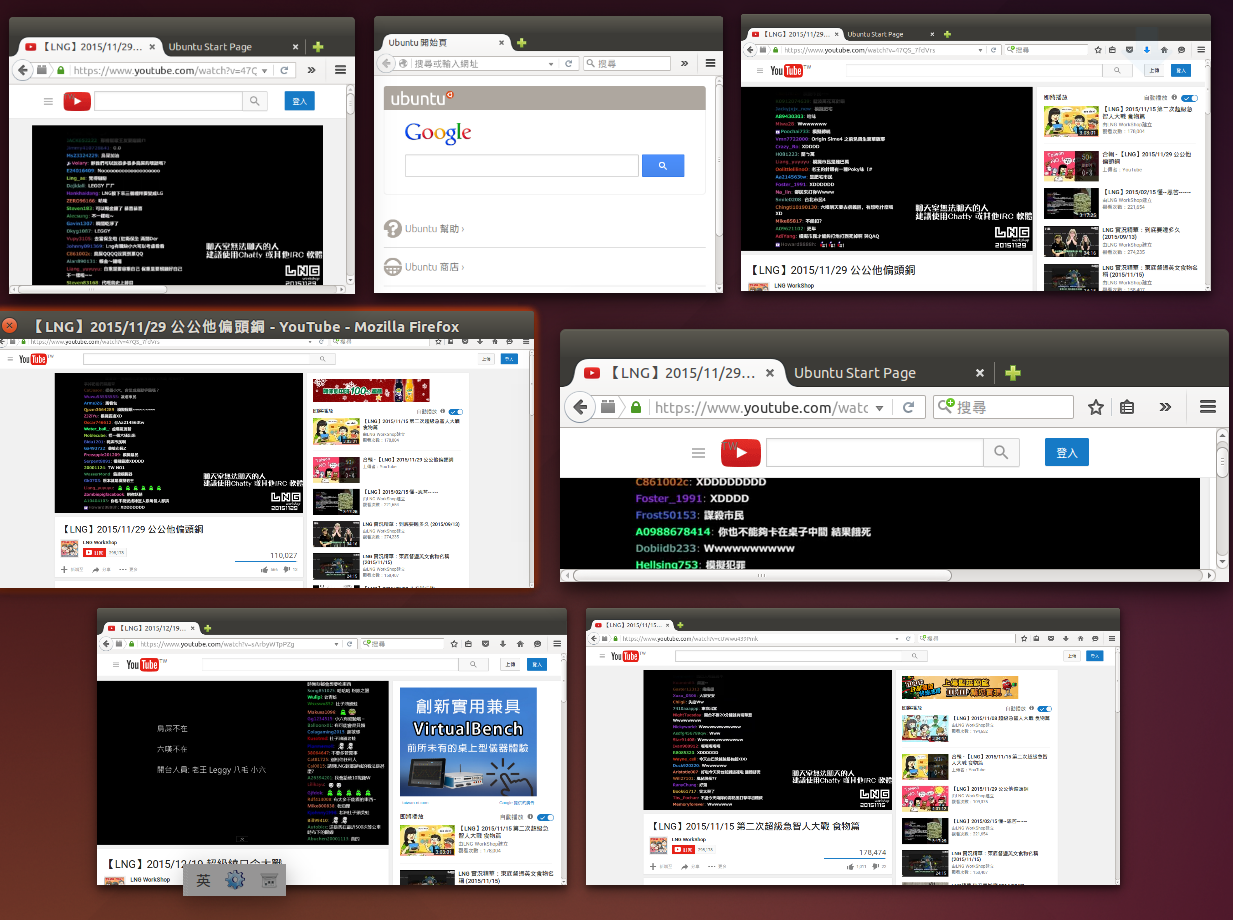
**最後透過字元陣列回傳虛擬區間跟實體區間還有虛擬量與實體量如下圖**



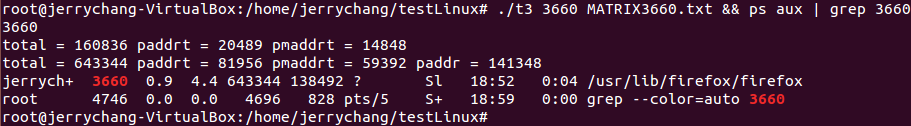


**3.2 partI**

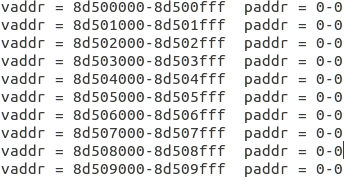
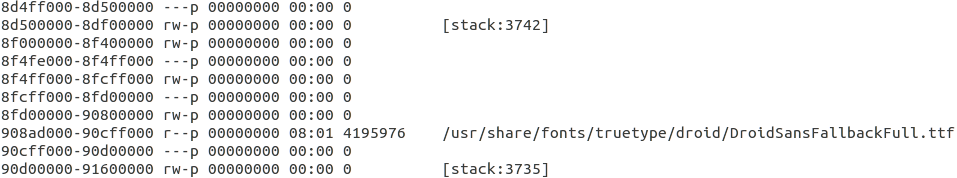
**開啟第一個firefox在圖中為jerrych+開啟的PID為3660，另外六個為root開啟的firefox –P的六個正在觀看youtube的firefox** 



**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。**

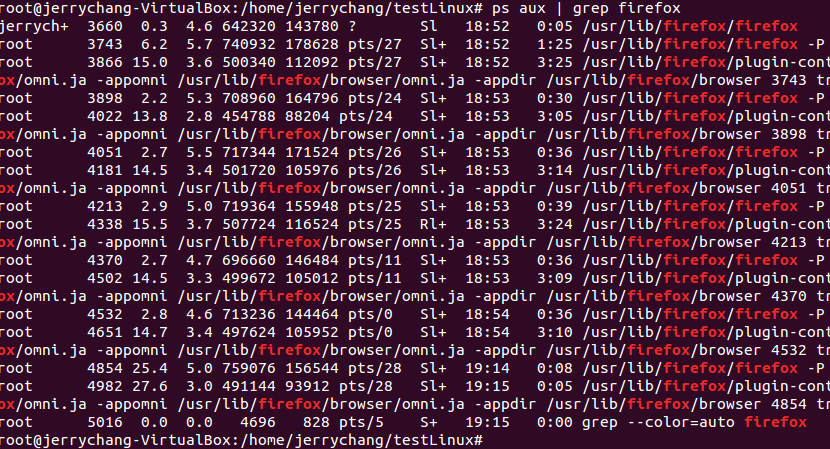


**從t3 test3.c這個test bench可以看出total virtual address量是643344，physical是 141348。所以百分比為141348/643344 約等於21.9%的虛擬記憶體有實體記憶體，從ps aux 指令可以看出虛擬記憶體為相同數量，實體則多了一點。而列出的虛擬記憶體與實體記憶體分配則寫成MATRIX3743.txt這個檔案，虛擬的區間是依照cat /proc/3660/maps > maps3660所列出的區間去列出，下圖為MATRIX3660.txt與maps3660的部分截圖。**

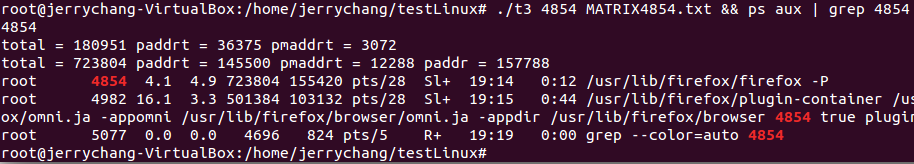
 

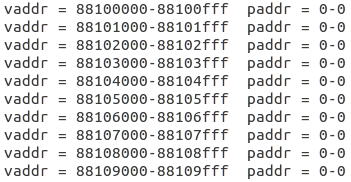
**4.Part II程式和執行結果說明**

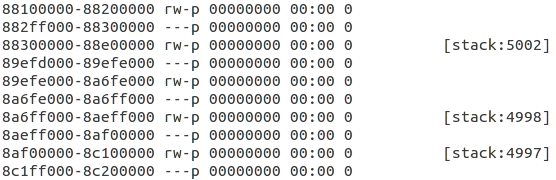
再新增一個firefox為process 2 並且也跑一個youtube影片，透過比較兩個ps aux | grep firefox結果找出process2的PID為5086。



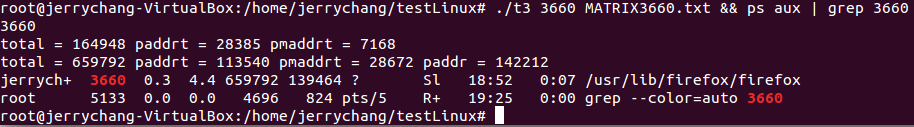
將process2的虛擬與實體記憶體一併記錄下來，可以從結果看出跟ps aux比較結果，虛擬記憶體量相同，實體記憶體稍微多一點點，再透過cat /proc/4854/maps > maps4854，與test bench(t3)所產生的MATRIX4854.txt做比較，而虛擬記憶體被轉成實體的比率為21.8%如下圖。

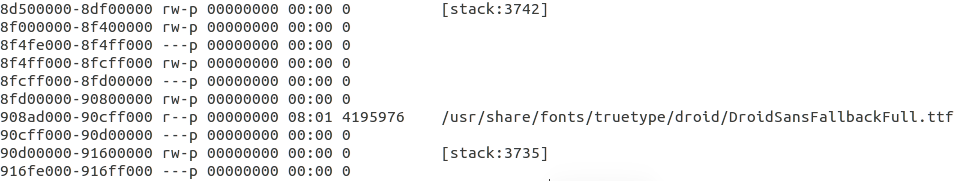
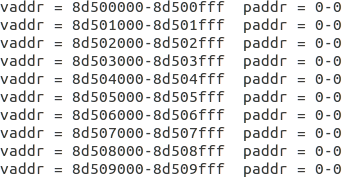






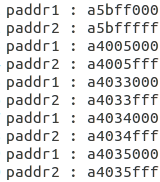
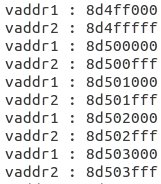
接著再跑一次process1的結果，從結果可以看出虛擬、實體記憶體多了一點點，比例下降為21.5%，比例下降如下圖，記憶體區間分配如下圖。





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。





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





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。



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;    } |