# OS Project2

Group 16
B06902068 洪晨翔

## 1. Programming Design

### In master.c:

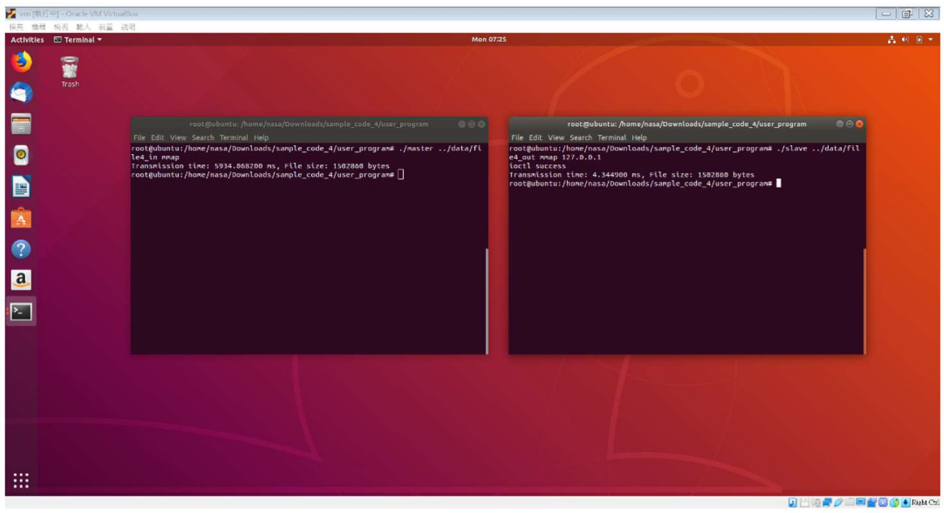
首先使用 mmap()，將整個 input 的 file 映射到一個給定的 address *addr。接著便可開始將 data 從*addr 寫到 device。我使用的方法是將 data 分成很多個 buffer，每次寫 BUF_SIZE 的 bytes 到 device 中，並檢查是否已到 data 的尾端，若已到尾端則只寫 data 剩下的 bytes 到 device 中。最後用 munmap()來取消 memory 的映射。


### In slave.c:

先將從 device 讀來的 data 全部保存到一個暫存的 buffer，同時確認 data 的大小，用 ftruncate()將 output file 的 size 調整至 data_size(原本是空檔案)。接著便可開 mmap()，將大小為 data_size 的 file 映射到一個給定的 address *addr。最後用 memcpy()，將 data 從 buffer 複製到 addr 中即可，並用 free()將 buffer 的 memory 釋放、用 munmap()來取消映射。另外在 written 所有 data 後，便可 call ioctl() 來與 device 溝通，取得 page descriptor 的相關資訊。

## 2. Result

開兩個 terminal，master 端執行./master ../data/fileX_in mmap，slave 端執行./slave ../data/fileX_out mmap 127.0.0.1，執行結果如下圖。另外，可以用 diff command 來確認 input file 和 output file 是一致的。dmesg 的部分，我在 slave_device.c 中多 call 了 pgd_val(), p4d_val(), pud_val(), pmd_val(), pte_val()來取得*pgd, *p4d, *pud, *pmd, *ptep 分別所在的 page table 所指向的 entry 中的值，也就是描述 page table entry 的幾種 struct pgd_t, p4d_t, pud_t, pmd_t, pte_t。可以觀察到 pte_val()的值和 source code 中的 pte 是相同的。
而實際上在 call mmap()時，空間的大小最好是 PAGE_SIZE 的整數倍，否則 memory space 會浪費掉，例如像是 file1，大小為 32bytes，這樣在 call mmap() 做出 virtual address 後，就有 PAGE_SIZE - 32 bytes 的空間沒有被使用到。



```
root@ubuntu:/home/nasa/Downloads/sample_code_4/data# diff file4_in file4_out
root@ubuntu:/home/nasa/Downloads/sample_code_4/data#
```

```
[ 2245.075199] aceept sockfd_cli = 0x00000000d89878bd
[ 2245.075201] got connected from : 127.0.0.1 49984
[ 2245.116434] slave device ioctl
[ 2245.116436] virtual address = 0x7f94f1bd1000
[ 2245.116437] pgd_value = 0x800000007603b067
[ 2245.116438] p4d_value = 0x800000007603b067
[ 2245.116438] pud_value = 0x79dc1067
[ 2245.116439] pmd_value = 0x7a528067
[ 2245.116439] pte_value = 0x80000000437c4867
[ 2245.116440] slave: 80000000437C4867
```

## 3. The comparison of the performance between file I/O and memory-mapped I/O

下面是針對四個 test data 用 mmap 和用 fcntl 的方法各跑 10 次的表格。(皆為先 call ./master 讓 data 傳到 device，再開./slave 讓 transmission time 較為精準。)
(單位: ms)

| type: fcntl | file1 | file2 | file3 | file4 |
|---|---|---|---|---|
| 1 | 0.0379 | 0.0432 | 0.2035 | 6.9597 |
| 2 | 0.0584 | 0.0419 | 0.2168 | 7.4655 |
| 3 | 0.0412 | 0.0476 | 0.1347 | 8.9046 |
| 4 | 0.0437 | 0.0618 | 0.1325 | 10.3745 |
| 5 | 0.0477 | 0.0628 | 0.1285 | 7.6481 |
| 6 | 0.0484 | 0.0445 | 0.1131 | 8.5075 |
| 7 | 0.0443 | 0.0511 | 0.1127 | 9.4611 |
| 8 | 0.0424 | 0.0441 | 0.1369 | 9.6878 |
| 9 | 0.0422 | 0.0759 | 0.1812 | 7.8543 |
| 10 | 0.0443 | 0.0365 | 0.1426 | 7.3383 |
| avg. | 0.04505 | 0.05094 | 0.15025 | 8.42014 |

| type: mmap | file1 | file2 | file3 | file4 |
|---|---|---|---|---|
| 1 | 0.0529 | 0.0502 | 0.1674 | 4.621 |
| 2 | 0.056 | 0.0713 | 0.0892 | 5.2198 |
| 3 | 0.0565 | 0.0471 | 0.1596 | 4.2515 |
| 4 | 0.0543 | 0.0663 | 0.1237 | 4.5261 |
| 5 | 0.0532 | 0.0628 | 0.1075 | 7.1746 |
| 6 | 0.04 | 0.0455 | 0.108 | 4.402 |
| 7 | 0.0453 | 0.0491 | 0.0922 | 7.7335 |
| 8 | 0.0482 | 0.0835 | 0.167 | 4.2441 |
| 9 | 0.0541 | 0.0687 | 0.1036 | 4.1291 |
| 10 | 0.0407 | 0.0604 | 0.1252 | 4.2176 |
| avg. | 0.05012 | 0.06049 | 0.12434 | 5.05193 |

1.理論上 mmap 是會比一般的 read/write I/O 快的，因為 mmap 直接把 file 的內容映射到 user space，後續直接對這塊空間進行操作，不需要重複的 system call (read(), write())，我們都知道 system call 因為來回於 user space 和 kernel 間，所以是非常慢的。
2.在小檔案 file1 和 file2 時，mmap 和 fcntl 的 transmission time 並沒有顯著分別，推測是因為檔案小，read() / write()的次數也少，system call()不會太多次，以 master 端為例，mmap 和 read/write I/O 都一樣是從 input file 複製到 memory space(只是方式不同而已)，再將 space 的 data 傳到 device，因此速度相近。
3.在大檔案 file3 和 file4 時，可以發現 transmission time 就有明顯的差距了，實際上扣掉從 slave_device read 來存到 memory space 的時間(兩種方式都有)，可能 fcntl 和 mmap 的 transmission time 相差更多，而且檔案越大，相差時間應該會更多，與理論相符。

## 4. Work list

B06902068 洪晨翔 100%

## 5. Reference

https://bit.ly/2WzZiQi
https://bit.ly/31mO0xx
https://bit.ly/2WotSHC
https://bit.ly/2WpKyyt