OS Project2 Report

一、組員

B07901069 劉奇聖
B05202050 劉安浚
B05901184 賴言禮
B07901184 陳映樵
B05103055 黃子瑋

網站：https://mortalhappiness.github.io/OSProject2/

二、設計

1. **Input parameter**
   Input parameter的部份我設計成和sample一樣，但是在parameter數目不對的時候會print usage出來，例如說有2個檔案輸入但卻只給一個檔名就會print出help message。

2. **參數設計**
   BUF_SIZE為512，MAP_SIZE為8192（兩個page大）。

3. **Master side**
   (a) user_program
   一開始會先把master device打開，如果method是mmap的話則將device mmap到一段MAP_SIZE大的記憶體，接著對於每個檔案計算檔案大小並把它們打開，並且為該檔案開一個socket接收來自slave的連接。如果method是fcntl的話就不斷read檔案並write到device；如果是mmap的話則不斷把檔案mmap到一段不大於MAP_SIZE的記憶體，並把該記憶體的內容copy到device的mmap，再call ioctl叫device把內容傳出去，直到所有檔案都傳出去為止。

   (b) master_device
   只要user_program call write就把message copy到一個buffer，再把該buffer的內容傳出去。如果user_program call mmap的話則allocate一塊MAP_SIZE大的記憶體並把該段記憶體的所有page設成reserved避免被swap out，再把該段記憶體map到user space，user_program call munmap再free掉該記憶體，當user_program call ioctl的時候就把該段記憶體的內容傳出去。

4. **Slave side**
   (a) user_program
   一開始會先把slave device打開，如果method是mmap的話則將device mmap到一段MAP_SIZE大的記憶體，接著對於每個檔案把檔案打開，為該檔案開一個socket連接到master。如果method是fcntl的話就不斷read device並write到file裡；如果是mmap的話則不斷call ioctl叫device把接收到的檔案放在device mmap裡，接著再根據devicereturn的value得知file需要增長多大，ftruncate並

mmap該file，最後再把device mmap的內容copy到file mmap，直到所有檔案都接收到為止。

(b) slave_device
只要user_program call read就從socket接收message到一個buffer，再把該buffer的內容copy到user space指定的buffer。如果user_program call mmap的話則allocate一塊MAP_SIZE大的記憶體並把該段記憶體的所有page設成reserved避免被swap out，再把該段記憶體map到user space，user_program call munmap再free掉該記憶體，當user_program call ioctl的時候就不斷從socket接收message直到填滿該段記憶體或沒有message接收，再return寫了多長的message到該段記憶體回user_program。

5. **測資設計**（位於input/ 資料夾下的所有檔案）
   (a) sample_input_1/
   助教提供的測資，裡面包含數個較小的純文字檔。
   (b) sample_input_2/
   助教提供的測資，裡面包含一個較大的純文字檔。
   (c) pdf/
   這個資料夾裏面放的都是pdf檔，想測試看看非純文字檔會不會傳成功。
   (d) medium_case/
   內含4個4096 bytes的檔案，想測試檔案大小恰好為一個page size大的時候的效能。
   (e) medium_case_off/
   內含4個4097 bytes的檔案，想測試檔案大小為比一個page size大一個byte的時候效能是否與 (d) 有差異。

三、file I/O 與 memory-mapped I/O 執行結果

在output資料夾中有output_file和time_size兩個資料夾，output_file中的檔案完全和input一樣（有diff過）。time_size的話對於每個測資有fnctl和mmap的兩種結果，每個檔案點開會有5行，此為我們對每組測資各跑五次以方便取平均和標準差做效能分析。（以下是在兩台不同virtual machine上各執行master和slave的結果）

以下的表格為各測資的執行時間平均及標準差，表格下方的兩張圖片為mmap時master端和slave端的page descriptors。

sample_input_1 (total size: 23462 bytes)		
(time:ms)	fcntl	mmap
mean	0.63496	0.5955
std	0.011313	0.017003

```
[ 7671.471548] [slave page descriptor] 01FFFF0000000800
```

```
[ 8410.073365] [master page descriptor] 01FFFF0000008900
```

sample_input_2 (total size: 12022885 bytes)		
(time: ms)	fcntl	mmap
mean	27.4627	11.7270
std	5.1692	0.46954

```
[ 7750.389875] [slave page descriptor] 01FFFF0000000800
[ 8434.380823] [master page descriptor] 01FFFF0000008900
```

pdf (total size: 2379921 bytes)		
(time: ms)	fcntl	mmap
mean	5.5733	2.5198
std	0.07567	0.04425

```
[ 7784.747173] [slave page descriptor] 01FFFF0000000800
[ 8458.769943] [master page descriptor] 01FFFF0000000800
```

medium_case (total size: 16384 bytes)		
(time: ms)	fcntl	mmap
mean	0.282	0.28908
std	0.008969	0.02352

```
[ 7825.311079] [slave page descriptor] 01FFFF0000008900
[ 8486.141690] [master page descriptor] 01FFFF0000000800
```

medium_case_off (total size: 16388 bytes)		
(time: ms)	fcntl	mmap
mean	0.2895	0.26266

std	0.01668	0.007343

```
[ 7871.988779] [slave page descriptor] 01FFFF0000008900
[ 8510.955343] [master page descriptor] 01FFFF0000000800
```

結果：

fcntl在實作上會需要大量的system call，而通常system call會較memory operation花上不少時間，所以我們可以預測mmap會比起fcntl還要有效率。而從我們的實驗結果可以看出，在檔案較大的時候（sample_input_2 and pdf）時，mmap相比fcntl速度為fcntl的2倍以上，而在檔案較小的情況之下（sample_input_1 and medium_case），兩者的效率差不多，我們猜測可能是因為在小檔案的情況下，mmap initialize的時候所產生的page fault以及連帶的更新TLB裡的資訊所花的時間會抵消mmap理論上對fcntl的優勢，所以才會表現出兩者效率沒有相差多少的情形。甚者，我們可以預測在file size很小的時候，fcntl的效率會比mmap好。

在stability方面，我們可以看到每一組執行五次時間的標準差都小於0.1，因此效能上很穩定，不會有一些很糟糕的情況發生（除了fcntl sample_input_2，可看到前三次都二十幾ms，後兩次三十幾ms，推測是傳大檔案時網路的穩定性問題，如後兩次突然queuing delay變大等）。

另外在medium_case和medium_case_off這兩個case，實驗後發現效率並無顯著差異。

四、file I/O 與 memory-mapped I/O 比較

mmap 優點：

1. 一般I/O需要將資料寫到buffer且存取速度慢。而mmap將資料映射到虛擬記憶體上，通過對這段記憶體的讀取和修改，實現對文件的讀取和修改，因為省去buffer這一層的緣故，mmap速度較快。

2. 通常呼叫一次system call的成本相對較高，因為CPU除了需要處理interrupt之外，還需要做context switch，因此成本相對一般function高。而mmap只有一開始的時候需要去分配記憶體之外，其他時候就可以交由user application去直接操作記憶體，相比read()或是write()需要讓系統一次又一次的去處理I/O，許多不必要的overhead可以去掉，因此mmap相比fcntl來說更有效率。

3. 呼叫fcntl的read()或write()這兩個system call時需要copy_from_user()或copy_to_user()，也就是要使data在kernel space以及user space之間轉移，然而因為mmap是直接讓user可以直接access到device的memory，因此這一層轉移在mmap就可以省略，因此較fcntl來說更省時間。

mmap 缺點：

1. mmap分配的記憶體空間都是以page size為最小單位，如此一來若將小檔案map在記憶體上，可能會有page使用率不高的問題，造成internal fragmentation。

2. 相比fcntl來說，mmap需要在device端額外實作，因此對於developer的負擔比較大。另外因為mmap在實作方面比較複雜，因此在檔案較小的情況下，fcntl這種比較簡單的方法可能會顯得較有效率。

結論：

雖然mmap可能會浪費比較多記憶體空間，或是在檔案較小時performance較差，但是在一般的情況下來說mmap的速度會比fcntl快上2~3倍，因此如果開發者有能力的話，基本上建議實作mmap，而使用者來說用mmap的效率會比一般的I/O高上不少。

五、組內分工

	劉奇聖	劉安浚	賴言禮	陳映樵	黃子瑋
比重	40 %	20 %	15 %	15 %	10 %

六、Reference

1. https://linux-kernel-labs.github.io/refs/heads/master/labs/memory_mapping.html?fbclid=IwAR1jZGBSMkknbMdXj45a6zPlwAlu3_0b-rxLxlvr613ZuONCD3VPWA5Rbu4
2. https://man7.org/linux/man-pages/man2/mmap.2.html
3. https://lwn.net/images/pdf/LDD3/ch15.pdf
4. http://rswiki.csie.org/lxr/http/source/include/linux/mm_types.h?v=linux-4.5.4#L44
5. https://www.man7.org/linux/man-pages/man2/ioctl.2.html
6. https://man7.org/linux/man-pages/man2/ftruncate.2.html