OS Project2 Report

模組設計

在這份作業中，我們被要求製作的是用於輸入輸出的字元裝置character device(cdev)，但根據給出的sample code裡面使用的其實是misc device(雜項裝置)。
基本的File I/O在sample code裡面已經做完大部分了，我們主要是照下面的順序去增加/修改struct裡面的成員來增加memory mapping的功能。

```
miscdevice -> fileops -> mmap
```

miscdevice

首先是miscdevice，主要結構如下：

```
struct miscdevice  {
        int minor;
        const char *name;
        const struct file_operations *fops;
        struct list_head list;
        struct device *parent;
        struct device *this_device;
        const struct attribute_group **groups;
        const char *nodename;
        umode_t mode;
};
```

我們只對一部分的成員作定義，剩下的不須更動。

```
static struct miscdevice master_dev = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = "master_device",
        .fops = &master_fops
};
```

file operations

之後對fops部分的結構做定義，一樣先看原本的file_operations的結構如何:

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ...
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        ...
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, un
        ...
} __randomize_layout;
```

對應的做出master_device/slave_device的master_fops/slave_fops。
其中read/write的實作利用了ksocket的krecv/ksend來達成。ioctl的部分也使用了krecv/ksend確保slave端永遠在master端完成I/O或是mmap之後才結束或是memcpy。
open/release的部分則是分別使用kmalloc/kfree處理用來存放Buffer的空間。

```
static struct file_operations master_fops = {
        .owner = THIS_MODULE,
        .unlocked_ioctl = master_ioctl,
        .open = master_open,
        .read = recv_msg,
        .write = send_msg,
        .release = master_close,
        .mmap = master_mmap
}; //similar for slave
```

mmap

最後是Sample code沒有實作到的memory mapping部分，一樣先看看對應mmap的結構：

```
struct vm_area_struct {
        /* The first cache line has the info for VMA tree walking. */
        unsigned long vm_start;               /* Our start address within vm_mm. */
        unsigned long vm_end;         /* The first byte after our end address
                                         within vm_mm. */
        ...

        /* Second cache line starts here. */

        struct mm_struct *vm_mm;      /* The address space we belong to. */
        pgprot_t vm_page_prot;        /* Access permissions of this VMA. */
        unsigned long vm_flags;               /* Flags, see mm.h. */
        ...
        /* Function pointers to deal with this struct. */
        const struct vm_operations_struct *vm_ops;
        ...
        void * vm_private_data;               /* was vm_pte (shared mem) */
        ...
} __randomize_layout;
```

這邊不管其他的函數，最直接的想法就是每次都使用ksocket在不同模組間傳輸資料，控制好同步後再利用memcpy將module中的private_data複製到output file。

```
static int master_mmap(struct file *filp, struct vm_area_struct *vma){
        remap_pfn_range(vma,
                vma->vm_start,
                virt_to_phys(filp->private_data) >> PAGE_SHIFT,
                vma->vm_end - vma->vm_start,
                vma->vm_page_prot
        );
        vma->vm_flags |= VM_RESERVED;
        return 0;
} //same in slave
```

User Program設計

在這裡我們確保了幾件事情：

1. 有一個方便的makefile可以一次將所有的模組以及UserProgram進行編譯
2. master及slave的執行先後順序不影響執行結果。
3. slave接收檔案的數量將"小於等於"master殘餘未發送的檔案數量
4. 在執行期間出錯的時候，使用者會得到精美的sterr提示
5. 即使是超過Buffer大小的檔案，程式仍然會正確的將檔案複製成功。

但仍然有些沒有被實作的功能例如不支援超過一個slave同時進行檔案傳輸。

大致可以將我們組的master端以及slave端進行的流程說明如下:

master端

1. 收集即將傳輸的檔案數，N_master
2. 檢查參數數量正確無誤
3. 確認傳輸的方法，fcntl/mmap
4. 打開master_device
   while(N_master!=0):
5. 創建socket
6. 傳送給slave殘餘的未傳輸檔案數，N_master
7. 收集來自slave的N_slave
8. 透過master_device傳輸N_slave個檔案給slave
9. 調整N_master <- N_master - N_slave
10. 關閉socket
    end while
11. 關閉master_device

slave端

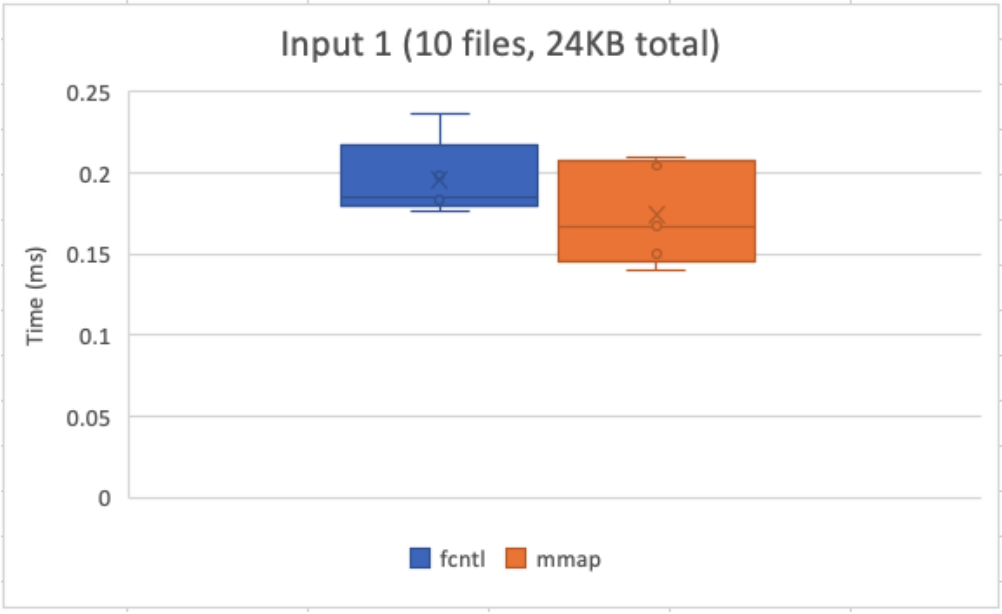
1. 收集即將傳輸的檔案數，N_slave
2. 檢查參數數量正確無誤
3. 確認傳輸的方法，fcntl/mmap
4. 打開slave_device
5. 創建socket
6. 收集來自master的未傳輸檔案數，N_master
7. 比較N_slave及N_master，調整N_slave後傳送給master
8. 透過slave_device接收N_slave個檔案並存檔
9. 關閉socket
10. 關閉slave_device

測試結果

- Input 1
  - input/sample_1/*
  - 10個檔案，共24KB

fcntl		1		2		3		4		5		
sample_1	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	32	0.0112	0.0186	0.0133	0.0228	0.0129	0.0225	0.0168	0.0216	0.0156	0.0194	
2	859	0.0228	0.0167	0.0302	0.0182	0.0301	0.0203	0.0217	0.0161	0.0246	0.016	
3	1,343	0.0199	0.0157	0.0199	0.0157	0.0219	0.0193	0.0162	0.0155	0.016	0.0137	
4	1,663	0.0272	0.0264	0.0179	0.0162	0.0217	0.0162	0.0167	0.0161	0.0156	0.014	
5	2,042	0.0368	0.037	0.0184	0.0173	0.017	0.0148	0.0165	0.016	0.0152	0.0138	
6	3,065	0.0273	0.0284	0.0166	0.0155	0.0173	0.0157	0.018	0.0175	0.016	0.0146	
7	2,808	0.0235	0.0199	0.0162	0.0151	0.0168	0.0154	0.0268	0.0263	0.0165	0.015	
8	4,056	0.0207	0.0204	0.0169	0.0154	0.0177	0.0162	0.0223	0.0215	0.0171	0.0158	
9	3,659	0.0215	0.0216	0.0236	0.0236	0.0179	0.0166	0.0219	0.0214	0.0232	0.022	
10	4,619	0.0277	0.0296	0.0198	0.0174	0.0192	0.0176	0.0244	0.0239	0.0229	0.0253	
total	24,146	0.2386	0.2343	0.1928	0.1772	0.1925	0.1746	0.2013	0.1959	0.1827	0.1696	
avg.			0.23645		0.185		0.18355		0.1986		0.17615	0.19595

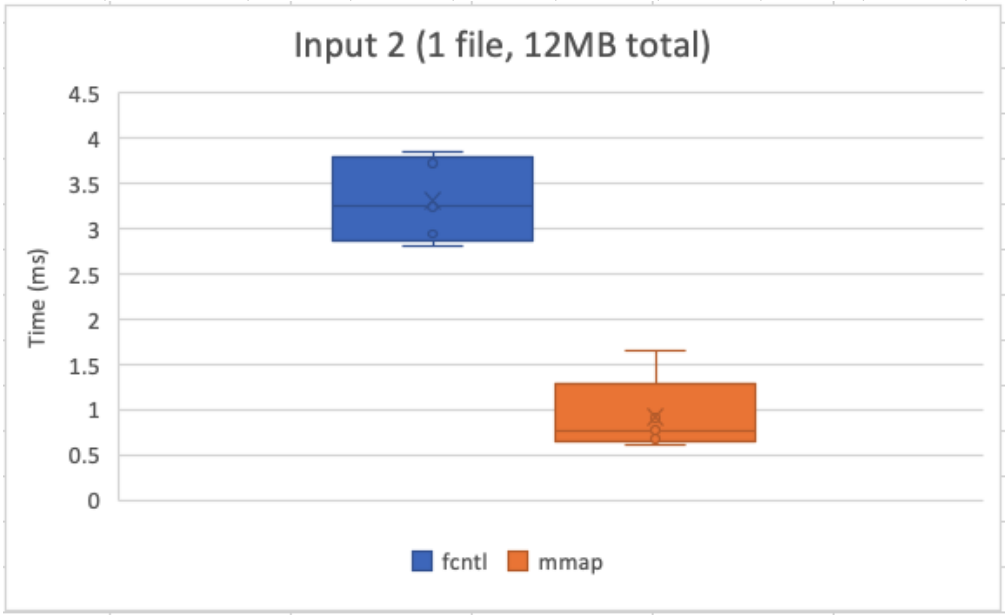
mmap		1		2		3		4		5		
sample_1	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	32	0.0155	0.0237	0.0121	0.0205	0.0121	0.0236	0.0192	0.0246	0.0122	0.0182	
2	859	0.0224	0.0167	0.0226	0.0205	0.0384	0.0315	0.0274	0.0235	0.0213	0.0189	
3	1,343	0.0145	0.0141	0.0192	0.016	0.0256	0.0259	0.0208	0.0178	0.0144	0.0137	
4	1,663	0.0137	0.0136	0.0144	0.014	0.0199	0.0193	0.0154	0.015	0.0126	0.012	
5	2,042	0.015	0.0155	0.014	0.0135	0.0184	0.018	0.0241	0.0247	0.0117	0.0117	
6	3,065	0.0228	0.0222	0.0134	0.0133	0.0186	0.0188	0.0226	0.0219	0.015	0.0158	
7	2,808	0.0184	0.0186	0.013	0.0132	0.019	0.0184	0.0262	0.026	0.0153	0.0153	
8	4,056	0.0143	0.0142	0.0135	0.0135	0.0189	0.0191	0.0181	0.018	0.0129	0.0123	
9	3,659	0.0135	0.0134	0.0134	0.0133	0.0185	0.0183	0.0163	0.0159	0.0116	0.0115	
10	4,619	0.0157	0.0163	0.0134	0.0138	0.0185	0.0189	0.015	0.0161	0.0116	0.0118	
total	24,146	0.1658	0.1683	0.149	0.1516	0.2079	0.2118	0.2051	0.2035	0.1386	0.1412	
avg.			0.16705		0.1503		0.20985		0.2043		0.1399	0.17428



- Input 2
  - input/sample_2/*
  - 1個檔案，共12MB

fcntl		1		2		3		4		5		
sample_1	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	12,022,885	3.5171	4.2007	2.0717	3.8167	1.9067	3.6987	3.1768	4.2854	2.4936	3.9936	
total	12,022,885	3.5171	4.2007	2.0717	3.8167	1.9067	3.6987	3.1768	4.2854	2.4936	3.9936	
avg.			3.8589		2.9442		2.8027		3.7311		3.2436	3.3161

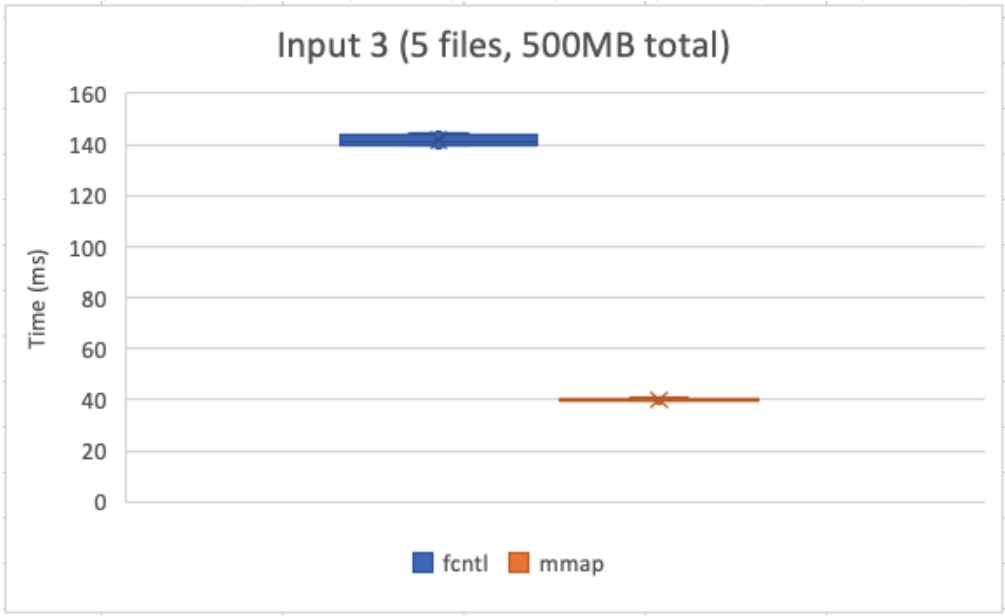
mmap		1		2		3		4		5		
sample_1	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	12,022,885	0.3907	0.82	0.4332	1.1027	0.4194	0.9299	0.5001	1.3135	0.9935	2.3093	
total	12,022,885	0.3907	0.82	0.4332	1.1027	0.4194	0.9299	0.5001	1.3135	0.9935	2.3093	
avg.			0.60535		0.76795		0.67465		0.9068		1.6514	0.92123



- Input 3
  - input/sample_3/*
  - 5個檔案，共500MB
  - 自行準備的測資

fcntl		1		2		3		4		5		
sample_3	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	103,809,024	26.43	28.54	26.81	28.22	26.15	27.96	25.48	27.63	25.83	27.81	
2	103,809,024	31.38	28.7	30.55	27.74	29.08	27.4	30.72	27.51	30.13	27.19	
3	103,809,024	30.58	27.66	31.38	27.76	30.48	27.31	29.51	27.24	30.02	27.49	
4	103,809,024	30.37	27.62	29.86	27.84	30.01	27.7	29.31	27.38	29.72	27.34	
5	103,809,024	30.4	27.62	29.52	27.48	29.18	27.24	27.76	27.32	27.65	27.04	
total	519,045,120	149.16	140.14	148.12	139.04	144.9	137.61	142.78	137.08	143.35	136.87	
avg.			144.65		143.58		141.255		139.93		140.11	141.905

mmap		1		2		3		4		5		
sample_3	file size (B)	master	slave	master	slave	master	slave	master	slave	master	slave	time (ms)
1	103,809,024	7.674	8.263	7.656	8.211	7.803	8.751	7.83	8.383	8.22	8.814	
2	103,809,024	7.64	7.649	8.184	8.191	8.371	7.979	7.546	7.567	7.754	7.785	
3	103,809,024	8.274	8.311	7.579	7.661	7.603	7.686	8.266	8.306	7.829	7.81	
4	103,809,024	7.712	7.732	8.334	8.266	7.771	7.704	8.386	8.322	7.806	7.79	
5	103,809,024	7.854	7.77	7.546	7.585	7.667	7.726	8.411	8.589	7.75	7.78	
total	519,045,120	39.154	39.725	39.299	39.914	39.215	39.846	40.439	41.167	39.359	39.979	
avg.			39.4395		39.6065		39.5305		40.803		39.669	39.8097



結果討論

我們會設計 Input 3 的想法是覺得 memory-mapped I/O 隨著檔案大小的增長，其效能優勢會越來越明顯。實驗的結果成功應證這點，Input 1 、 2 、 3 的總檔案大小是越來越大的，而執行時間差距也越來越明顯。我們認為原因如同上課講到的， mmap 在讀寫時只做記憶體複製，減少了 disk I/O 的次數，所以執行時間較短。

組內分工

姓名	學號	分工內容
許傑盛	b04901003	實作 device driver mmap 相關功能
謝至宥	b04501002	實作 user program, 報告撰寫
阮明皓	b05901062	實作 user program, 報告撰寫
徐乃威	r08922137	實作 user program, 報告撰寫

Reference

1. 深入理解Linux內核 3/e
2. Linux Device Driver 3/e
3. Linux Kernel（6）- miscdev (http://nano-chicken.blogspot.com/2009/12/linux-modules6-miscdev.html)
4. cdev及三大重要的struct (https://blog.xuite.net/tzeng015/twblog/113271907-cdev%E5%8F%8A%E4%B8%89%E5%A4%A7%E9%87%8D%E8%A6%81%E7%9A%84struct)
5. 內核驅動mmap Handler利用技術 (https://r00tk1ts.github.io/2017/12/30/Kernel%20Driver%20mmap%20Handler%20Exploitation/)
6. Linux内存管理实践-使用fault()实现内存映射 (http://edsionte.com/techblog/archives/3993)
7. Why there isn't a munmap callback in struct file_operation? (https://stackoverflow.com/questions/25098592/why-there-isnt-a-munmap-callback-in-struct-file-operation)
8. LinuxKernel miscdevice v4.14.25 (https://elixir.bootlin.com/linux/v4.14.25/source/include/linux/miscdevice.h#L64)
9. LinuxKernel vm_operation_struct v4.14.25 (https://elixir.bootlin.com/linux/v4.14.25/source/include/linux/mm.h#L367)
10. LinuxKernel file_operations v4.14.25 (https://elixir.bootlin.com/linux/v4.14.25/source/include/linux/fs.h#L1692)