OS Project2 Report

組員:

b07902002 連崇安(程式測試,20%)

b07902007 伍柏豪(程式撰寫,20%)

b07902012 江宗翰(程式撰寫,20%)

b07902076 許世儒(程式撰寫,20%)

b07902117 陳漱宇(report撰寫,20%)

➢ 設計

master → master_device → slave_device → slave

❑ master.c

```
while(file_size[j] - offset > map_sz) {
        file_address = mmap(NULL, map_sz, PROT_READ, MAP_SHARED, file_fd[j], offset);
        //perror("before kernel_address");
        kernel_address = mmap(NULL, map_sz, PROT_WRITE, MAP_SHARED, dev_fd, offset);
        //perror("after kernel_address");

        offset += map_sz;
        //perror("before memcpy\n");

        memcpy(kernel_address, file_address, map_sz);
        //perror("after memcpy\n");
        ioctl(dev_fd, 0x12345678, map_sz);
        munmap(file_address, map_sz);
        munmap(kernel_address, map_sz);
}
```

用 mmap 分別 map 到 input file 和 master_device, 接著利用 memcpy 將 file 記憶體的內容複製到 master_device 的記憶體。

❑ master_device.c

```
case master_IOCTL_MMAP:
        ret = ksend(sockfd_cli, file->private_data, ioctl_param, 0);
        break;
```

當 master 將 data 複製到 master_device 的記憶體後, master_divice 會利用 ksend 將資料送給 slave_device。

❑ slave_device.c

```
case slave_IOCTL_MMAP:
        ret = krecv(sockfd_cli, file->private_data, PAGE_SIZE, MSG_WAITALL);
        break;
```

slave_device 會利用 krecv 負責接收從 master_device 傳來的訊息。

❑ slave.c

```
while ((ret = ioctl(dev_fd, 0x12345678)) > 0) {
        while(ret == 0 && file_size[j] == 0)
                ret = ioctl(dev_fd, 0x12345678);
        ftruncate(file_fd[j], offset+ret);
        file_address = mmap(NULL, ret, PROT_WRITE, MAP_SHARED, file_fd[j], offset);
        kernel_address = mmap(NULL, ret, PROT_READ, MAP_SHARED, dev_fd, offset);
        memcpy(file_address, kernel_address, ret);
        munmap(file_address, ret);
        munmap(kernel_address, ret);
        offset += ret;
        file_size[j] += ret;
}
```

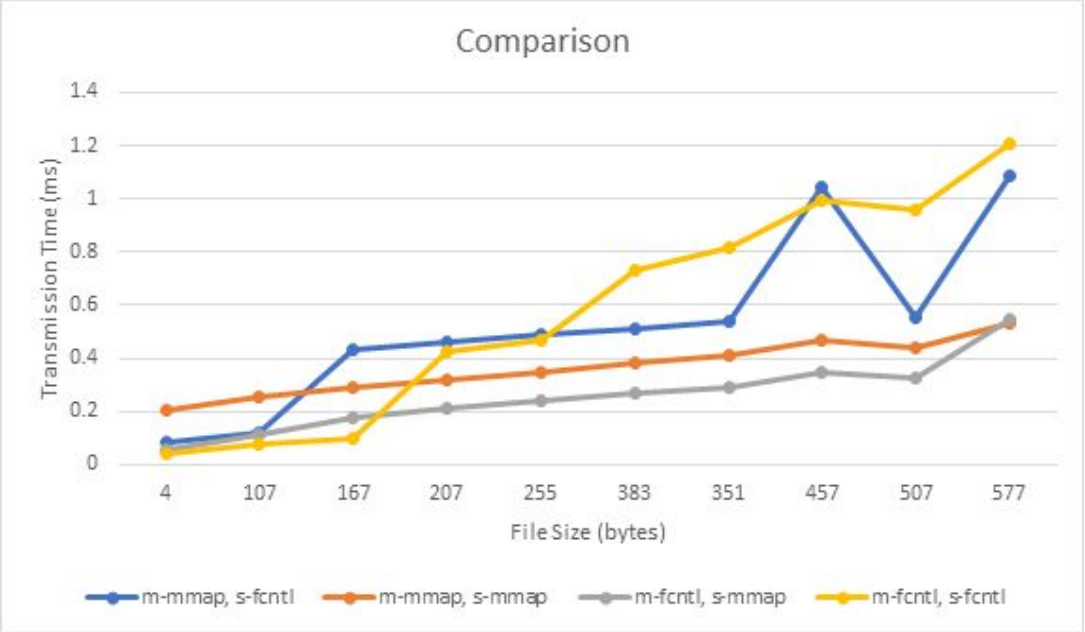
將檔案用 ftruncate 調整成合適的大小, 接著用 mmap 分別 map 到 output file 和 slave_device, 再用 memcpy 將 slave_device 記憶體的內容複製到 output file 的記憶體。

➢ 系統

linux 4.14.25

➢ file I/O 和 memory-mapped I/O 的結果與效能差異

下圖為實際測試 sample_input_1 的結果(m:master,s:slave):



在小檔案下, file I/O (fcntl) 與 memory-mapped I/O (mmap) 差異並不大, 因為雖然 memory-mapped I/O 是直接進行修改 virtual memory 進而實現檔案的修改, 但因為 mmap 本身也會有一些 overhead, 所以在小檔案上並不見得會顯著地比 file I/O 快。但是在大檔案的情形下, 就能夠看出來 memory-mapped I/O 會明顯地比 file I/O 快。

➢ 自訂 input (our_input.txt)

我們設定此檔案的大小為 4096 * 100 bytes, 而我們執行一次 mmap 所要的 memory 大小為 4096 * 50 bytes, 恰好為 50 個 pages (1 page = 4096 bytes), 因此剛好只需要執行兩次 mmap 即可。

➢ bonus

master_device

```
#ifdef ASYN
static struct workqueue_struct *wq_fcntl;
DECLARE_WORK(work_fcntl, (void *)send_msg);
static struct workqueue_struct *wq_mmap;
DECLARE_WORK(work_mmap, (void *)master_ioctl);
#endif
```

```
#ifdef ASYN
wq_mmap = create_workqueue("master_wq_mmap");
queue_work(wq_mmap, &work_mmap);
#endif
```

```
#ifdef ASYN
wq_fcntl = create_workqueue("master_wq_fcntl");
queue_work(wq_fcntl, &work_fcntl);
#endif
```

slave_device

```
#ifdef ASYN
static struct workqueue_struct *wq_fcntl;
DECLARE_WORK(work_fcntl, (void *)receive_msg);
static struct workqueue_struct *wq_mmap;
DECLARE_WORK(work_mmap, (void *)slave_ioctl);
#endif
```

```
#ifdef ASYN
wq_map = create_workqueue("slave_wq_mmap");
queue_work(wq_mmap, &work_mmap);
#endif
```

```
#ifdef ASYN
wq_fcntl = create_workqueue("slave_wq_fcntl");
#endif
```

master_device 和 slave_device 各自利用 workqueue 的方式將工作排程進去，之後再依序執行 workqueue 中的工作即可達到 asynchronous 的效果。

測試結果

➔ our_input.txt(File Size: 4096 * 100 bytes)

- synchronous: Transmission Time 5.67 ms
- asynchronous: Transmission Time 4.459 ms

➢ reference

1. 映射在啓動時使用remap_pfn_range用戶空間reservever內存
   http://hk.uwenku.com/question/p-sxbrlzgx-qo.html?fbclid=IwAR3YPIpHOMJtRVIjXslV78oz1ka__nbxBJWjS5zLc3hYI85A7N7rOBbNMzA
2. linux-kernel - 使用 kmalloc mmap: 映射分配的內核緩衝區
   https://hant-kb.kutu66.com/linux/post_1958601?fbclid=IwAR3GHVNx9POyq95no1qYJ_nDxgoePE8xwuvX7sQc2eeQmUwXi4KaDzJjkXE
3. Linux核心工作佇列

   https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/609894/?fbclid=IwAR37GYwm8_cITAlHq7tOZTwZxDgnb5EPGnTUBPof4neZDyAyzgM5wZlp_5s
4. ftruncate-linux man page

   https://linux.die.net/man/2/ftruncate
5. Concurrency Managed Workqueue

   https://www.kernel.org/doc/html/v4.14/core-api/workqueue.html?fbclid=IwAR3fbExbt1Gai5LDcJ774X-Hw1RifTexFiazsnIHAqIQagrmkkfkzD1IjHI
6. 有與 Regular Group 37 討論