

OS Project 2 Report--Group 29

Design

這是一個 Master-Slave 架構，我們需要讓 Master 以及 Slave device 都支援 mmap。以下分別說明四份程式碼：

user_program/master.c:

```
case 'm': //mmap : memcpy() and mmap()
    dev_addr = mmap(NULL, NPAGE * BUF_SIZE, PROT_WRITE, MAP_SHARED, dev_fd, 0);
    HANDLE_FATAL(dev_addr, "Can't mmap to master device!");
    *dev_addr=0;
    while(offset < file_size) {
        if(offset + length > file_size)
            length = file_size - offset;
        file_addr = mmap(NULL, length, PROT_READ, MAP_SHARED, file_fd, offset);
        HANDLE_FATAL(file_addr, "Can't mmap to file!");

        memcpy(dev_addr, file_addr, length);
        ret = ioctl(dev_fd, 0x12345678, length);
        HANDLE_FATAL(ret, "ioctl server sending error");

        munmap(file_addr, length);
        offset += length;
    }
    break;
    ioctl(dev_fd, 0x1234567a, dev_addr);
    munmap(dev_addr, NPAGE * PAGE_SIZE);
}
```

將檔案 map 到 user_program 記憶體後，再將 device 記憶體 map 到 user_program，而後利用 memcpy 將檔案拷貝至該 map 記憶體，再透過 ioctl 通知 device mapping 已完成。

master_device/master_device.c:

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

抓到 user_program 中 master.c 傳送的通知後，找到記憶體資料，再透過 ksend 傳送記憶體資料給 slave device。

user_program/slave.c:

```

case 'm': // mmap : mmap()/memcpy()
    dev_addr = mmap(NULL, NPAGE * PAGE_SIZE, PROT_READ, MAP_SHARED, dev_fd, 0);
    HANDLE_FATAL(dev_addr, "Can't mmap to slave device!");
    while((ret = ioctl(dev_fd, 0x12345678)) > 0)
    {
        while(ret == 0 && file_size == 0)
            ret = ioctl(dev_fd, 0x12345678);

        length = ret;
        ftruncate(file_fd, offset+length);
        file_addr = mmap(NULL, length, PROT_WRITE, MAP_SHARED, file_fd, offset);
        HANDLE_FATAL(file_addr, "Can't mmap to file!");
        memcpy(file_addr, dev_addr, length);

        munmap(file_addr, length);
        offset += length;
        file_size += length;
    }
    break;

```

```

if (dev_addr){ // There's a memory region mapping to the device
    ioctl(dev_fd, 0x1234567a, dev_addr);
    munmap(dev_addr, NPAGE * PAGE_SIZE);
}

```

Slave device 讀到檔案後，slave program 使用 mmap 以及 memcpy 寫道 output file，傳 page 的 address 去 printk
slave_device/slave_device.c:

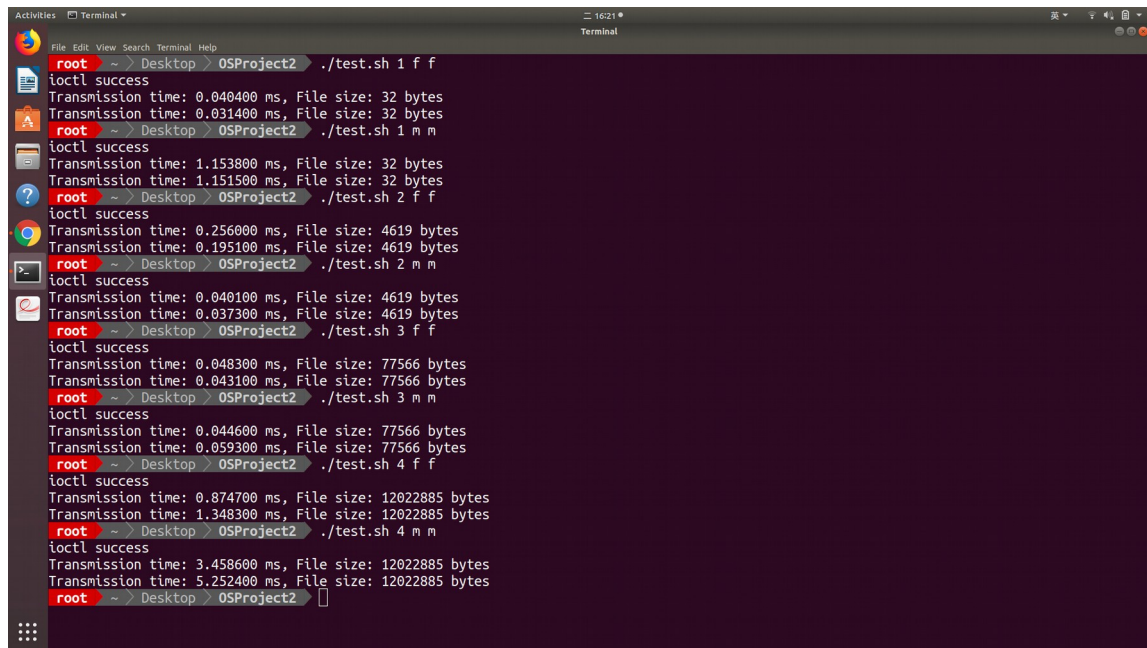
```

case slave_IOCTL_MMAP:
    recv_n = krecv(sockfd_cli, file->private_data, sizeof(buf), 0);
    ret = recv_n;
    break;

```

連線後取得資料，將 device 處記憶體 map 至 user_program，再將內容 map 到將輸出的檔案

Test Case:

A terminal window titled 'Terminal' showing a series of test results. The user is in the directory ~/Desktop/OSProject2. The tests are performed using ./test.sh with arguments 1 f f, 1 m m, 2 f f, 2 m m, 3 f f, 3 m m, 4 f f, and 4 m m. Each test shows 'ioctl success' and transmission details: 'Transmission time: [value] ms, File size: [value] bytes'. For file I/O (f), transmission times are very low (e.g., 0.040400 ms for 32 bytes). For memory-mapped I/O (m), transmission times are significantly higher (e.g., 1.153800 ms for 32 bytes, 3.458600 ms for 12022885 bytes).

```
root ~ > Desktop > OSProject2 > ./test.sh 1 f f
ioctl success
Transmission time: 0.040400 ms, File size: 32 bytes
root ~ > Desktop > OSProject2 > ./test.sh 1 m m
ioctl success
Transmission time: 1.153800 ms, File size: 32 bytes
root ~ > Desktop > OSProject2 > ./test.sh 2 f f
ioctl success
Transmission time: 0.256000 ms, File size: 4619 bytes
root ~ > Desktop > OSProject2 > ./test.sh 2 m m
ioctl success
Transmission time: 0.040100 ms, File size: 4619 bytes
root ~ > Desktop > OSProject2 > ./test.sh 3 f f
ioctl success
Transmission time: 0.048300 ms, File size: 77566 bytes
root ~ > Desktop > OSProject2 > ./test.sh 3 m m
ioctl success
Transmission time: 0.044600 ms, File size: 77566 bytes
root ~ > Desktop > OSProject2 > ./test.sh 4 f f
ioctl success
Transmission time: 0.874700 ms, File size: 12022885 bytes
root ~ > Desktop > OSProject2 > ./test.sh 4 m m
ioctl success
Transmission time: 3.458600 ms, File size: 12022885 bytes
```

Result analysis and compare file I/O & memory-mapped I/O:

若是小檔案如 file1_in、file2_in，執行的時間小於 0.1ms 時，m/m 並沒有明顯的優勢，但隨著檔案越來越大，兩者逐漸拉開的差距，說明了 mmap I/O 在大檔案時，確實能夠提高傳遞效率。

Master side:

檔案漸大時，Master 方使用會比使用 fcntl 快。

Reason:

mmap 時呼叫 system call 的次數較少。

Slave side:

Slave 方使用 mmap 與 fcntl 的時間快慢無固定。

Reason:

接收檔案無法事先得知大小，導致得事後壓縮，因此不一定。

組員貢獻:

Slave, Master: B06902051,B05902132,B06902042

Bonus: B06902117,B06902065

Report: B06902051,B05902132,B06902042

Bonus_Report: B06902117,B06902065