2020 OS Project 2
Synchronous Virtual Device

資工三 b06902009 柯建宇
資工三 b06902023 林恩廷
資工三 b06902115 袁才育
資工三 b06902119 張原豪
資工三 b06902126 劉羽忻

1. Design

- Device
  - 使用助教提供之sample code進行修改，並在 master_device 和 slave_device 中增加 mmap 的部分。
  - 實作呼叫 mmap 時所需要的函式(master_mmap / slave_mmap)，透過 remap_pfn_mmap 將 device 的 memory 與 mmap 空間建立連結。

在master_device.c 和 slave_device.c 中加入支援mmap的open和close

```
void mmap_open(struct vm_area_struct *vma) {}
void mmap_close(struct vm_area_struct *vma) {}

struct vm_operations_struct mmap_vm_ops = {
    .open = mmap_open,
    .close = mmap_close
};
```

master_device.c 中的 master_mmap()、ksend()

```
static int master_mmap(struct file *file, struct vm_area_struct *vas);

static int master_mmap(struct file *file, struct vm_area_struct *vas){
    if (remap_pfn_range(vas, vas->vm_start, vas->vm_pgoff,
        vas->vm_end - vas->vm_start, vas->vm_page_prot))
        return -EIO;
    vas->vm_flags |= VM_RESERVED;
    vas->vm_private_data = file->private_data;
    vas->vm_ops = &mmap_vm_ops;
    mmap_open(vas);
    return 0;
}

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

slave_device.c 中的 slave_mmap()、krecv()

```
static int slave_mmap(struct file *file, struct vm_area_struct *vas);

static int slave_mmap(struct file *file, struct vm_area_struct *vas){
    if (remap_pfn_range(vas, vas->vm_start, vas->vm_pgoff,
        vas->vm_end - vas->vm_start, vas->vm_page_prot))
        return -EIO;
    vas->vm_flags |= VM_RESERVED;
    vas->vm_private_data = file->private_data;
    vas->vm_ops = &mmap_vm_ops;
    mmap_open(vas);
    return 0;
}

case slave_IOCTL_MMAP:
    ret = krecv(sockfd_cli, file->private_data, PAGE_SIZE, 0);
```

- Master
  - 當使用fcntl模式時，直接從檔案進行input，並寫到device。
  - 當使用mmap模式時，首先設定一個固定的map size，並利用mmap()設定檔案位址和kernel位址，接著直接把檔案位址memcpy()到kernel位址，直到整個檔案傳輸完畢為止。
  - 其中第二個ioctl() command中的0x12345676僅作default用途。

master.c 中 mmap的case

```
case 'm': //mmap
    while (offset < file_size) {
        size_t transfer_size = min(MAP_SIZE, file_size - offset);
        // printf("transfer_size : %lu\n", transfer_size);

        if((file_address = mmap(NULL, transfer_size, PROT_READ,
            MAP_SHARED, file_fd, offset)) == MAP_FAILED) {
            perror("mapping input file");
            return 1;
        }
        if((kernel_address = mmap(NULL, transfer_size, PROT_WRITE,
            MAP_SHARED, dev_fd, offset)) == MAP_FAILED) {
            perror("mapping output device");
            return 1;
        }

        memcpy(kernel_address, file_address, transfer_size);
        offset += transfer_size;
        ioctl(dev_fd, 0x12345678, transfer_size);
        ioctl(dev_fd, 0x12345676, (unsigned long)file_address);
        munmap(file_address, transfer_size);
    }
    break;
```

- Slave
  - 當使用fcntl模式時，直接從device進行input，並寫到檔案中。
  - 當使用mmap模式時，利用mmap()設定檔案位址和kernel位址，接著直接把kernel位址memcpy()到檔案位址，直到整個檔案傳輸完畢為止。

slave.c 中 mmap的case

```
case 'm':
    while(ret = ioctl(dev_fd, 0x12345678) > 0) {
        if(posix_fallocate(file_fd, offset, ret) != 0) {
            perror("posix_fallocate error");
            return 1;
        }
        file_address = mmap(NULL, ret, PROT_WRITE, MAP_SHARED, file_fd, offset);
        kernel_address = mmap(NULL, ret, PROT_READ, MAP_SHARED, dev_fd, offset);
        memcpy(file_address, kernel_address, ret);
        offset += ret;
    }
    file_size = offset;

    break;
```

- 生成測試資料的程式
  - 由於助教所提供的input檔案皆不大，估計不容易在實驗結果中看出fcntl與mmap之間傳輸資料速度的差異，因此我們決定設計出一個.c檔，用於產生接近100MB的大檔案來做input的測試資料。

test.c 中用於生成測資的程式碼

```
#include <stdio.h>
int main(){
    FILE *fp;
    char str1[] = "group35 data\n";
    char str2[] = "OS SOS plz\n";
    fp = fopen("newdata.txt", "w");
    for (int i = 0; i < 4000000; i++){
        fputs(str1, fp);
        fputs(str2, fp);
    }
    fclose(fp);
    return 0;
}
```

2. Result

- File 1: number of file = 10, total size = 24 KB
- File 2: number of file = 1, total size = 12 MB
- File 3: number of file = 1, total size = 96 MB
- 除了fcntl to fcntl 和 mmap to mmap，我們也額外比較了 fcntl to mmap 與 mmap to fcntl 的 performance

Trail 1:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**
File 1	6.670300 ms	8.276700 ms	5.003900 ms	5.895800 ms
File 2	6937.718100 ms	7992.302700 ms	5996.698200 ms	8007.385600 ms
File 3	35974.774000 ms	34915.401300 ms	20013.891600 ms	16029.832500 ms

Trail 2:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**
File 1	6.952500 ms	5.966500 ms	8.796500 ms	6.971200 ms
File 2	6982.892300 ms	6998.108300 ms	6958.137100 ms	7956.897800 ms
File 3	42945.070100 ms	31991.367900 ms	22025.747500 ms	17996.517900 ms

Trail 3:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**
File 1	10.371700 ms	14.159500 ms	7.229700 ms	7.822700 ms
File 2	6987.190300 ms	4075.063800 ms	8038.412200 ms	6998.797300 ms
File 3	35011.328600 ms	24006.274900 ms	19035.278900 ms	17999.467500 ms

Trail 4:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**
File 1	6.575600 ms	7.318200 ms	5.291300 ms	14.601500 ms
File 2	6955.088100 ms	7027.337300 ms	8998.444300 ms	7998.283000 ms
File 3	29051.525600 ms	27970.991400 ms	21578.427300 ms	17992.634400 ms

Trail 5:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**

File 1	8.389000 ms	6.449500 ms	7.253700 ms	16.476000 ms
File 2	7960.112700 ms	6973.554000 ms	7987.220500 ms	9927.326400 ms
File 3	40014.410600 ms	25979.151300 ms	22304.289300 ms	14965.663200 ms

Average:

	fcntl to fcntl	**fcntl to mmap**	**mmap to fcntl**	**mmap to mmap**
File 1	7.79182 ms	8.43408 ms	6.71502 ms	10.35344 ms
File 2	7164.6003 ms	6613.27322 ms	7595.78246 ms	8177.73802 ms
File 3	36599.42178 ms	28972.63736 ms	20991.52692 ms	16996.8231 ms

3. Discussion & Conclusion

- 比較：
  - 在檔案較小的情況下做測試時，不管是用mmap或fcntl傳送或接收， 所做出來的transmission time的差異都不算太大。然而在file size較大的情況下，mmap的傳輸速度有非常大的進步。依照傳輸時間排序大約是：
    fcntl/fcntl > fcntl/mmap > mmap/fcntl > mmap/mmap
  - 小檔案傳輸時間相近的原因我們認為應該與 mmap 背後的實作方式有關。進行 mmap I/O時，kernel 需要建立 page table 和 TLB，倘若發生 TLB miss 或 page fault，kernel 也需要花費大量時間進行 memory access，造成最後傳輸時間與 file I/O 相近。
  - 大檔案傳輸時間有較明顯差距的原因在於 mmap I/O 是把 kernel space 的 memory 進行一次性的複製到 user space，相較於 file I/O 減少了許多讀寫的 overhead。因此，在檔案越大的情況下，mmap會有比較明顯的優勢。

4. Bonus

- 這個部分由於時間上的關係我們並沒有進行實作。但經過一些 research 後我們覺得 asynchronous 的傳送時間應該會比 synchronous 來的短，因為 synchronous 要等 buffer 裝滿後才進行 I/O，而 asynchronous 卻不需要。buffer 的大小對於結果也會有一定的影響，若 buffer 太小，則差距應該不會太明顯。

5. Contributions of Team Members

小組成員	負責內容	分工比重
資工三 柯建宇 b06902009	整理mmap與fcntl使用方法、設計input、report撰寫	20%
資工三 林恩廷 b06902023	設計master與slave架構、實作 master 與 slave	20%
資工三 袁才育 b06902115	整理mmap與fcntl使用方法、設計input、report撰寫	20%
資工三 張原豪 b06902119	編譯kernel、實作 master 與 slave	20%
資工三 劉羽忻 b06902126	整理mmap與fcntl使用方法、設計input、report撰寫	20%

6. References

1. remap_pfn_range: https://wooyun.js.org/drops/%E7%AE%80%E5%8D%95%E7%B2%97%E6%9A%B4%E6%9C%89%E6%95%88%E7%9A%84mmap%E4%B8%8Eremap_pfn_range.html
2. mmap: https://man7.org/linux/man-pages/man2/mmap.2.html
3. Code reference: https://github.com/andy920262/OS2016/tree/master/project2