OS Project 2 Report - Regular Group 4

設計

(含說明自己程式的設計針對自己準備的測試檔設計的好處，我們期待有創意的方向，在好的設計的同時，越有自己的想法分數會越高，切勿抄襲他人想法)

device

我們先加入mmap至master_device與slave_device中

```
static int my_mmap(struct file *filp, struct vm_area_struct *vma);
void mmap_open(struct vm_area_struct *vmarea){}
void mmap_close(struct vm_area_struct *vmarea){}
```

file operations中也加入mmap

```
static struct file_operations master_fops = {
...
//add for mmap
.mmap = my_mmap
};
```

然後

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

提供vm之相關操作，open/close函式與mmap之函式。

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

master_device

master_IOCTL_CREATESOCK 會製造一個socket並產生連結。

```
static long master_ioctl(struct file *file, unsigned int ioctl_num, unsigned lo
{
...
case master_IOCTL_MMAP:
ret = ksend(sockfd_cli, file->private_data, ioctl_param, 0);
break;
...
}
```

slave_device

在slave_device中，slave_IOCTL_CREATESOCK 會產生與master端的Ksocket連結。

```
static long slave_ioctl(struct file *file, unsigned int ioctl_num, unsigned lon
{
    ...
    case slave_IOCTL_MMAP:
                while (offset < PAGE_NUM * PAGE_SIZE){
                        rev = krecv(sockfd_cli, file->private_data + offset, BUF
                        if (rev == 0){
                                break;
                        }
                        offset += rev;
                }
                ret = offset;
                break;
    ...
}
```

Master

每次在讀取前我們將input file與kernel socket的buffer 以mmap映射至process之virtual memory，再從input file中將我們的map的大小 (PAGE_NUM * PAGE_SIZE) 的data複製至kernel socket的buffer。並重複直行至EOF。

Slave

我們使用ioctl與mmap，使slave device之kernel buffer接收來自master device之kernel buffer的資料。若ret(ioctl之回傳值)非0，我們用mmap將slave device buffer與output file映射至virtual memory，並從slave device buffer複製ret大小之資料至output file之map。而若ret為0，表示資料讀完了，跳出讀取之迴圈。

page descriptor

我們印出每次傳送檔案時，device之第一個memory address。

```
[  212.820400] master : 8000000000000267
[  212.821207] slave : 8000000000000227
```

比較

比較file I/O 和 memory-mapped I/O 的結果與效能差異，並解釋

mmap 對文件的讀取操作跨過了page cache，減少數據的拷貝次數，用記憶體讀寫取代I/O讀寫，而作業系統則負責disk flush 的時機，如此可減少disk I/O的次數與對應等待時間。
雖然在某些情況下也會產生page fault ，但是mmap 不再需要從磁盤中複製文件過來，而可直接使用已經保存在記憶體中的文件數據（少了一次複製buffer的動作）
綜合上述兩點，mmap 比file IO更加快速。
因此在有大量數據需要處理時，mmap/mmap的傳輸時間比functl/functl有效率許多。
然而在小檔案處理方面，則因為總時間不長，造成各模式間的傳輸時間差異不明顯。甚至mmap因為有大量overhead的原因（建立page table 與TLB flush）而可能會稍微慢一些。

分別以fcntl及mmap方式傳送target_file_1得到的結果

```
root@wryyy-VirtualBox:~/Desktop/sample_code_for_4.14.25/user_program# ./slave 1
target_file_1_out fcntl 127.0.0.1
Transmission time: 0.085800 ms, Total file size: 32 bytes
root@wryyy-VirtualBox:~/Desktop/sample_code_for_4.14.25/user_program# ./slave 1
target_file_1_out mmap 127.0.0.1
Transmission time: 0.092900 ms, Total file size: 32 bytes
```

分別以fcntl及mmap方式傳送target_file得到的結果

```
root@wryyy-VirtualBox:~/Desktop/sample_code_for_4.14.25/user_program# ./slave 1
fout mmap 127.0.0.1
Transmission time: 4.082600 ms, Total file size: 12022885 bytes
root@wryyy-VirtualBox:~/Desktop/sample_code_for_4.14.25/user_program# ./slave 1
fout fcntl 127.0.0.1
Transmission time: 4.707400 ms, Total file size: 12022885 bytes
root@wryyy-VirtualBox:~/Desktop/sample_code_for_4.14.25/user_program#
```

分工

組內分工表及分工比重，為個人分數的依據

學號	名字	分工
B07902116	陳富春	寫程式
B07902069	李哲宇	report
B06902052	張集貴	debug, 拍 demo 影片

reference

說明程式碼的參考來源或與其他組別討論

https://github.com/b05902046/OS-Project-2/blob/master/report.pdf (https://github.com/b05902046/OS-Project-2/blob/master/report.pdf)