2020 OS Project 2

- Synchronous Virtual Device Report

資工二 B06902079 陳柏瑋

資工二 B07902057 董函

化學四 B05203017 許晉洋

化學四 B05203008 廖哲宏

Programming Design

Device

master_device.c

master_device.c中增加的mmap部分

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

static const struct vm_operations_struct master_vm_ops = {
    .open = mmap_open,
    .close = mmap_close,
};

static int master_mmap(struct file *file, struct vm_area_struct *vma){
    remap_pfn_range(vma, vma->vm_start, virt_to_phys(file->private_data) >> PAGE_SHIFT,
                    vma->vm_end - vma->vm_start, vma->vm_page_prot);
    vma->vm_ops = &master_vm_ops;
    vma->vm_flags |= VM_RESERVED;
    vma->vm_private_data = file->private_data;
    mmap_open(vma);
    return 0;
}
```

並且在master_fops{}中增加下列部分，以連結到master_mmap()

```
    .mmap = master_mmap
```

並將master_open(),master_close()改成如下，以正確的在開始和結束的時候分配和收回memory

```c
int master_close(struct inode *inode, struct file *filp)
{
    kfree(filp->private_data);
    return 0;
}

int master_open(struct inode *inode, struct file *filp)
{
    filp->private_data = kmalloc(MAP_SIZE, GFP_KERNEL);
    return 0;
}
```

最後當case為mmap時，使用ksend將檔案傳出

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

slave_device.c

和master_device.c類似，增加mmap部分

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

static const struct vm_operations_struct slave_vm_ops = {
    .open = mmap_open,
    .close = mmap_close,
    //.fault = mmap_fault
};

static int slave_mmap(struct file *file, struct vm_area_struct *vma){
    remap_pfn_range(vma, vma->vm_start, virt_to_phys(file->private_data) >>
PAGE_SHIFT,
                    vma->vm_end - vma->vm_start, vma->vm_page_prot);
    vma->vm_ops = &slave_vm_ops;
    vma->vm_flags |= VM_RESERVED;
    vma->vm_private_data = file->private_data;
    mmap_open(vma);
    return 0;
}
```

並且在slave_fops{}中增加下列部分，以連結到slave_mmap()

```c
    .mmap = slave_mmap
```

並將slave_open(),slave_close()改成如下，以正確的在開始和結束的時候分配和收回memory

```c
int slave_close(struct inode *inode, struct file *filp)
{
    kfree(filp->private_data);
    return 0;
}

int slave_open(struct inode *inode, struct file *filp)
{
    filp->private_data = kmalloc(MAP_SIZE, GFP_KERNEL);
    return 0;
}
```

最後當case為mmap時，使用krecv來接收檔案，並用memcpy在buf跟file之間移動。此外，為了避免overflow，slave_device從socket讀之前會先檢查offset+buffer是否會超出MAP_SIZE。

```c
        case slave_IOCTL_MMAP:
            offset = 0;
            while(1)
            {
                len = krecv(sockfd_cli, buf, sizeof(buf), 0);
                //printk("slave ioctl received %s byte", len);
                buf[len] = '\0';
                //printk("slave ioctl received: [%s]", buf);
                if (len == 0)
                {
                    break;
                }
                memcpy(file->private_data + offset, buf, len);
                //printk("file->private_data: [%s]", file->private_data);
                //printk("write received data to file->private_data
done\n");
                offset += len;
                if (offset + sizeof(buf) > MAP_SIZE)
                {
                    break;
                }
            }
            ret = offset;
            break;

```

User Program

master.c

master.c會先在argv中獲得總共要傳送的檔案數**N**，再開始一個重複**N**次的迴圈。

每次迴圈master會開啟一個新的socket,並和slave端建立連結，在mmap的部分，會先將offset，也就是傳送的起點設為0，並開始傳送，使用mmap將file_fd代表的檔案map到記憶體中(file_address)，再用memcpy()將之寫入dev_fd代表的檔案之中(kernel_address)，最後再使用munmap()解除map以歸還memory。詳細的mmap部分程式碼如下：

```c
    case 'm':
        map_length = MAP_SIZE;
        offset = 0;

```

```
        while(offset < file_size){
            if (file_size - offset < map_length){
                map_length = file_size - offset;
            }
            file_address = mmap(NULL, map_length, PROT_READ, MAP_SHARED, file_fd, offset);
            kernel_address = mmap(NULL, map_length, PROT_WRITE, MAP_SHARED, dev_fd, offset);
            memcpy(kernel_address, file_address, map_length);
            munmap(file_address, map_length);
            munmap(kernel_address, map_length);
            offset += map_length;
            ioctl(dev_fd, 0x12345678, map_length);
        }
        break;
```

slave.c

slave.c會先在argv中獲得總共要傳送的檔案數**N**，再開始一個重複**N**次的迴圈。

與master類似的，slave在每次迴圈都會先跟master建立連結再開始傳送。同樣先將offset設為0，再開始接受檔案的while迴圈，ret代表每次迴圈接受的大小。先利用posix_fallocate()預先分配磁碟空間，再使用mmap建立file_fd跟file_address，dev_fd跟kernel_address之間的mapping關係，並用memcpy()傳送檔案，最後再使用munmap()解除mapping關係，歸還memory。詳細的mmap部分程式碼如下：

```
    case 'm':
        printf("method: mmap\n");
        offset = 0;
        while(1)
        {
            ret = ioctl(dev_fd, 0x12345678);
            if (ret == 0){
                file_size = offset;
                break;
            }
            //printf("ioctl return: %d\n", ret);
            //printf("calling file mmap\n");
            posix_fallocate(file_fd, offset, ret);
            file_address = mmap(NULL, ret, PROT_WRITE, MAP_SHARED, file_fd, offset);
            //printf("calling dev mmap\n");
            kernel_address = mmap(NULL, ret, PROT_READ, MAP_SHARED, dev_fd, offset);
            //printf("memcpy...\n");
            // for (int i =0; i < 20; i++){
            //  printf(":%c",kernel_address[i]);
            // }

            memcpy(file_address, kernel_address, ret);
            //printf("calling file munmap\n");
            munmap(file_address, ret);
            //printf("calling file munmap\n");
            munmap(kernel_address, ret);

            offset += ret;
        }
        break;
```

Result

測試的部分我們使用了兩組data，第一組(Input1)是10個小檔案，大小大約都在KB等級，而第二組(Input2)是1個大檔案，大小則是為12MB左右。我們分別都進行了10次實驗，並去除掉電腦出現問題時的outlier，再取平均以避免單次實驗所產生的誤差。

以下是我們的實驗結果，X軸為Method，前面是master所使用的Methodd，後面則是slave所使用的Method，例如MF指的便是master使用mmap而slave使用fcntl。Y軸則是實驗的編號，總共進行了10次。時間單位為ms(毫秒，millisecond)

Input1 Result

Input1	**Master**				**Slave**			
	MM	MF	FM	FF	MM	MF	FM	FF
1	0.7604	0.2126	0.3363	0.2755	0.578	0.427	0.4449	0.2798
2	0.4437	0.2197	0.3507	0.2929	0.266	0.2744	0.4746	0.1396
3	0.2764	0.1999	0.4057	0.1672	0.2468	0.4571	0.3038	0.3592
4	0.2477	0.2806	0.5166	0.2911	0.3144	0.3377	0.6429	0.4219
5	0.2459	0.2542	0.3746	0.2601	0.2216	0.2782	0.2514	0.3946
6	0.2156	0.2477	0.5295	0.2638	0.223	0.2915	0.4307	0.2653
7	0.1928	0.3638	0.5766	0.3796	0.2217	0.2243	0.6344	0.376
8	0.5012	0.2946	0.2858	0.2763	0.2867	0.302	0.2665	0.2778
9	0.2735	0.2248	0.2385	0.3531	0.216	0.2165	0.3258	0.331
10	0.2745	0.1917	0.3488	0.2577	0.1627	0.3319	0.2197	0.2903
Avg.	0.34317	0.24896	0.39631	0.28173	0.27369	0.31406	0.39947	0.31355

Input2 Result

Input2	Master				Slave			
	MM	MF	FM	FF	MM	MF	FM	FF
1	2.893	1.921	2.2164	2.4483	2.9409	4.4322	2.1854	4.8201
2	0.7173	2.0595	2.8301	2.8845	1.6814	4.6915	2.754	4.4382
3	0.4167	1.6372	2.4364	2.1324	1.4147	4.6635	2.2392	4.5081
4	0.4037	1.7458	2.1598	2.6817	1.6349	4.3033	2.3519	4.6326
5	0.4735	1.837	2.2047	2.279	1.5676	4.0454	2.1825	4.6422
6	0.4275	2.7269	902.1921	2.3228	1.5348	4.9213	902.3038	4.4802
7	0.4414	1.9175	2.3496	1.9438	1.4765	4.6041	2.5946	4.3829
8	0.5924	901.348	2.3393	2.1012	1.2172	904.5886	2.2954	4.4054
9	0.5512	1.9336	2.389	2.1396	1.6195	4.2552	2.3833	4.583
10	0.8793	2.1186	2.3552	4.4314	1.7121	4.4829	2.4831	6.755
Avg.	0.7796	91.9245	92.3472	2.53647	1.67996	94.4988	92.3773	4.7647
Avg. w/o outlier	0.7796	1.9885	2.3645	2.53647	1.67996	4.4888	2.3854	4.7647

Page descriptor

Master:mmap/Slave:mmap

```
[ 6447.671142] master: 8000000039280267
[ 6447.671389] slave device ioctl
[ 6447.671393] slave: 8000000039300225
```

Master:mmap/Slave:fcntl

```
[ 6576.324973] master: 800000000C480267
```

Master:fcntl/Slave:mmap

```
[ 6599.483560] slave: 800000000C480225
```

Conclusion

Input1	Master				Slave			
	MM	MF	FM	FF	MM	MF	FM	FF
Avg.	0.34317	0.24896	0.39631	0.28173	0.27369	0.31406	0.39947	0.31355

Input2	Master				Slave			
	MM	MF	FM	FF	MM	MF	FM	FF
Avg.	0.7796	1.9885	2.3645	2.53647	1.67996	4.4888	2.3854	4.7647

從我們的實驗中可以得出以下結論，首先是在檔案大小不大時，從上面的Input1之結果可見，mmap所耗費的時間和fcntl所耗費的時間並沒有顯著差異；而當檔案大小較大時，從上面的Input2之結果可見，我們可以觀察到mmap能夠較為明顯的提升效率，大致呈現$MM > MF \simeq FM \geq FF$ 的趨勢。

而在比較slave跟master的差異時，我們可以得到以下結論，slave通常比會較慢，因為其要等待socket收到資料，並且即使網路速度很快，收的也無法太快，因為其一次只能夠收BUF_SIZE的大小，所以slave通常會較master慢上一些。

Reference

1. https://github.com/yang611/OS_project2
2. https://github.com/wangyenjen/OS-Project-2

Contribution

Name	Contribution	Percentage
資工二 B06902079 陳柏瑋	master, report	30%
資工二 B07902057 董函	參與討論	10%
化學四 B05203017 許晉洋	device driver, 實驗, demo	40%
化學四 B05203008 廖哲宏	slave, 協助debug	20%