

2020 OS Project 2 Report

1. Design

Master

We added mmap and multiple file transmission in user_program/master.c

- **Multiple file transmission**

When we transmit a file, we send the size of the file first. In this way, slave can know where the file ends and receives another file.

```
for(i = 0 ; i < N ; i++){
    strncpy(file_name, argv[2+i], sizeof(file_name)); // filepath
    if( (file_fd = open(file_name, O_RDWR)) < 0 ){
        perror("failed to open input file\n");
        return 1;
    }

    if( (file_size = get_filesize(file_name)) < 0 ){
        perror("failed to get filesize\n");
        return 1;
    }
    total_filesize += file_size;

    char size[512];
    sprintf(size, "%zu", file_size);
    write(dev_fd, size, 512); // tell the reciever how large is the file

    switch(method[0]){
        case 'f': //fcntl : read()/write()
            ...
            break;
        case 'm': //mmap
            ...
            break;
    }
}
```

- **mmap**

We mapped every 4096 bytes onto a memory page and got its address. Then we used this address to write data to master device.

```

for(ret = 0 ; ret < file_size ; ret += PAGE_SIZE){
    if( (file_address = mmap(NULL, PAGE_SIZE, PROT_READ, MAP_SHARED,
                             file_fd, ret)) == (void *) -1){
        perror("failed to map input file\n");
        return 1;
    }

    for(int j = 0; j < PAGE_SIZE && j < (file_size - ret); j += BUF_SIZE)
        write(dev_fd, &file_address[j], BUF_SIZE);
    ioctl(dev_fd, 0x12345676, (unsigned long)file_address);
    munmap(file_address, PAGE_SIZE);
}

```

Slave

We added mmap and multiple file transmission in user_program/slave.c

- **Multiple file transmission**

When we transmit a file, we receive the size of the file first. In this way, slave knows where the file ends.

```

for(int i = 0; i < N; i++){
    strcpy(file_name, argv[i+2]);
    if( (file_fd = open (file_name, O_RDWR | O_CREAT | O_TRUNC)) < 0)
    {
        perror("failed to open input file\n");
        return 1;
    }

    write(1, "ioctl success\n", 14);
    size_t remain;
    ret=read(dev_fd, buf, BUF_SIZE);
    sscanf(buf, "%zu", &remain);
    file_size = remain;
    total_filesize += file_size;

    switch(method[0])
    {
        case 'f'://fcntl : read()/write()
            ...
            break;
        case 'm'://mmap
            ...
            break;
    }
    close(file_fd);
}

```

- **mmap**

Everytime when we mapped 4096 bytes onto a memory page and got its address, we could write data to this address and unmapped the page. In addition, OS would transmit the content of the unmapped page to the file in disk. We repeated these steps until the transmission of all files finished.

```

char *mfile;
int cursor = 0;

while(remain > 0){
    if(remain > BUF_SIZE){
        read(dev_fd, buf, sizeof(buf));
        mfile = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
                     file_fd, (cursor/PAGE_SIZE)*PAGE_SIZE);
        memcpy(&mfile[cursor%PAGE_SIZE], buf, BUF_SIZE);
        cursor += BUF_SIZE;
        munmap(mfile, PAGE_SIZE);
        remain -= BUF_SIZE;
    }else{
        read(dev_fd, buf, remain);
        mfile = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
                     file_fd, (cursor/PAGE_SIZE)*PAGE_SIZE);
        memcpy(&mfile[cursor%PAGE_SIZE], buf, remain);
        cursor += remain;
        ioctl(dev_fd, 0x12345676, (unsigned long)mfile);
        munmap(mfile, PAGE_SIZE);
        remain = 0;
    }
}

```

2. Analysis

- Page descriptors master mmap / slave fcntl

[3691.345268] master: 800000006AC3B025

- Page descriptors master fcntl / slave mmap

[3598.071181] slave: 80000000621EF867

- Page descriptors master mmap / slave mmap

[3481.865682] master: 800000006AC3B025
[3481.865895] slave device ioctl
[3481.865897] slave: 8000000065603867

Time Analysis

- We transmitted 5 different size of files and estimate the average master transmission time:
 - A really small file (4 bytes).
 - A file slightly larger than buffer size (520 bytes).
 - A file slightly smaller than page size (4000 bytes).
 - A file slightly larger than page size (4200 bytes).
 - A really large file (100000 bytes).
- mmap to mmap

File size (bytes)	4	520	4000	4200	100000
Average time (ms)	0.0016	0.0046	0.0069	0.0240	5.9400

- fcntl to fcntl

File size (bytes)	4	520	4000	4200	100000
Average time (ms)	0.0012	0.0024	0.0100	0.0220	10.020

- mmap to fcntl

File size (bytes)	4	520	4000	4200	100000
Average time (ms)	0.0024	0.0065	0.0080	0.0120	4.9600

- fcntl to mmap

File size (bytes)	4	520	4000	4200	100000
Average time (ms)	0.0010	0.0018	0.0061	0.0170	7.9900

We can see that when the file size is very small (e.g. 4 bytes), fcntl's transmission time will be smaller than that using mmap.

Nonetheless, when the file size becomes larger. Mmap's transmission time will be smaller than fcntl's one day. In our experiment, we can observe this trend when the file size is 100000 bytes.

The reason why fcntl is faster than mmap in small size is because of the overhead in the memory mapped I/O, including the construction of TLB table and page fault.

When the size of the data is larger than the page size, fcntl needs to wait for I/O time, while mmap can simply move data to memory and continually process other tasks.

3. Member & Work

Department	Student ID	Name	Work
CSIE Junior	B06902039	賈本皓	slave.c
CSIE Junior	B06902053	張維哲	analysis
CSIE Junior	B06902067	許育銘	master.c
CSIE Junior	B06902069	許博翔	master.c
CSIE Junior	B06902128	鄭力誠	slave.c

4. Reference

- socket programming: <https://www.geeksforgeeks.org/socket-programming-cc/>
- mmap: <https://welkinchen.pixnet.net/blog/post/41312211-%E8%A8%98%E6%86%B6%E9%AB%94%E6%98%A0%E5%B0%84%E5%87%BD%E6%95%B8-mmap-%E7%9A%84%E4%BD%BF%E7%94%A8%E6%96%B9%E6%B3%95>