NTU CSIE 108-2 OS Project #2

組別：薛智文到我家
資工二 B07902096 李佳恩
資工二 B07902144 彭約博
數學三 B06201018 張芷榕
資工三 B06902131 吳冠穎
資工三 B06902075 林詩敏
資工三 B06902136 賴冠毓

1. 設計

(1) user program

① master(讀需要 mapping 的檔案，然後寫給 master device)

比起 sample code，加入了可以一次 I/O 多個檔案，以及 memory-mapped I/O 的功能，還有最後 output 的 Transmission time & File size 由單個檔案變成所 I/O 的檔案的總和。

a. 多個檔案的部分沒有什麼特別的，使用一個變數 number_of_file 去紀錄要處理的檔案數量，之後在 for 迴圈依序 I/O 要處理的檔案。

```
strcpy(number_of_file, argv[1]); // get the N (but in chars)

for (int i = 0; i < strlen(argv[1]); i++) // convert from chars to int
{
    num_of_file *= 10;
    num_of_file += argv[1][i] - '0';
}
char file_name[num_of_file + 3][50];
for (int i = 0; i < num_of_file; i++)
{
    strcpy(file_name[i], argv[i + 2]); // get the N file names
}

for (int i = 0; i < num_of_file; i++)
{
```

b. 而 memory-mapped I/O 的部分，先透過 mmap 得到一塊 memory(圖中的 kernelMemory)。再來進入迴圈，先計算還有多少資料需要 mapping，之後一樣透過 mmap 取得檔案的 memory(圖中的 mappedMemory)，使用 memcpy 將 mappedMemory 的 file_fd 資料 mapping 到 kernelMemory，之後用 munmap 釋放 mappedMemory，並使用 ioctl 通知此次 I/O 是否完成，如此重複直到迴圈結束(該檔案 mapping 完成)。最後用 ioctl 將 kernelMemory 中所

mapping 到的檔案資料寫入 dev_fd，然後釋放 kernelMemory 還給系統。

```
case 'm':
    kernelMemory = mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dev_fd, 0);
    for (int j = 0; j * MAP_SIZE < file_size; j++)
    {
        tmp = file_size - j * MAP_SIZE;
        if (tmp > MAP_SIZE)
            tmp = MAP_SIZE;
        mappedMemory = mmap(NULL, tmp, PROT_READ, MAP_SHARED, file_fd, j * MAP_SIZE);
        memcpy(kernelMemory, mappedMemory, tmp);
        munmap(mappedMemory, tmp);
        while (ioctl(dev_fd, master_IOCTL_MMAP, tmp) < 0 && errno == EAGAIN)
            ;
    }
    if (ioctl(dev_fd, 0x111, kernelMemory) == -1)
    {
        perror("ioclt server error\n");
        return 1;
    }
    munmap(kernelMemory, MAP_SIZE);
    break;
```

c. Transmission time & File size 的部分則透過 total_transmissionTime & total_file_size 這兩個變數來記錄。

```
    gettimeofday(&end, NULL);
    transmissionTime = (end.tv_sec - start.tv_sec) * 1000 + (end.tv_usec - start.tv_usec) * 0.0001;
    //printf("Transmission time: %lf ms, File size: %lu bytes\n", transmissionTime, file_size);
    total_transmissionTime += transmissionTime;
    total_file_size += file_size;

    close(file_fd);
    close(dev_fd);
}
printf("Transmission time: %lf ms, File size: %lu bytes\n", total_transmissionTime, total_file_size);
```

② slave(從 slave device 讀資料，然後寫出 output 檔)

與 master 類似，一樣加入了可以一次 I/O 多個檔案、memory-mapped I/O 以及加總 Transmission time & File size 的功能，只是在 slave 端 coding style 有所不同。

a. 多個檔案的部分與 master 一樣，不再贅述。

```
strcpy(number_of_file, argv[1]); // get the N (but in chars)

for (int i = 0; i < strlen(argv[1]); i++)
{ // convert from chars to int
    num_of_file *= 10;
    num_of_file += argv[1][i] - '0';
}

char filename[num_of_file + 3][50];
for (int i = 0; i < num_of_file; i++)
{
    strcpy(filename[i], argv[i + 2]); // get the N file names
}

strcpy(method, argv[num_of_file + 2]);
strcpy(ip, argv[num_of_file + 3]);

for (int i = 0; i < num_of_file; i++)
{
```

b. memory-mapped I/O 的部分也類似 master，只是流程反過來。一樣先透過 mmap 得到一塊 memory(圖中的 kernel_mem)。再來進入迴圈，先使用 ioctl 判斷此次 I/O 是否完成，若已完成代表 mapping 結束，若沒有則繼續做 mapping，之後分別用 offset & offlen 紀錄當前 I/O 的位置&要讀進來的資料長度，然後一樣透過 mmap 取得我們 output 檔案的 memory(圖中的 mapped_mem)，使用 memcpy 將 kernel_mem 的 dev_fd 資料 mapping 到 mapped_mem，之後用 munmap 釋放 mappedMemory，如此重複到迴圈結束(該檔案 mapping 完成)。最後用 ftruncate 更新我們 output 出的檔案大小，然後釋放 kernel_mem 還給系統。

```c
else if (strcmp(method, "mmap") == 0)
{ // if method is mmap
    kernel_mem = mmap(NULL, MAPSZ, PROT_READ, MAP_SHARED, dev_fd, 0);
    while (1)
    {
        while ((val = ioctl(dev_fd, slave_IOCTL_MMAP)) < 0 && errno == EAGAIN)
            ;

        if (val < 0)
        {
            ioctlerror();
            return 1;
        }
        else if (val == 0)
            break;

        posix_fallocate(fd, fileSize, val);
        size_t offset = (fileSize / PAGESZ);
        offset *= PAGESZ;
        size_t offlen = fileSize;
        offlen -= offset;

        mapped_mem = mmap(NULL, offlen + val, PROT_WRITE, MAP_SHARED, fd, offset);
        memcpy(mapped_mem + offlen, kernel_mem, val);
        munmap(mapped_mem, offlen + val);
        fileSize += val;
    }
    ftruncate(fd, fileSize);

    int ok2 = checkerror2();
    if (ok2 == 1)
        return 1;
    munmap(kernel_mem, MAPSZ);
    //break;
}
```

```c
void ioctlerror()
{
    fprintf(stderr, "ioctl error\n");
    return;
}
```

```c
int checkerror3()
{
    if (ioctl(dev_fd, slave_IOCTL_CREATESOCK, ip) == -1)
    { //slave_IOCTL_CREATESOCK : connect to master in the device
        fprintf(stderr, "ioctl create slave socket error\n");
        return 1;
    }
    return 0;
}
```

c. Transmission time & File size 的部分也與 master 相同，不再贅

述。

```
    gettimeofday(&end, NULL);
    transmissionTime = (end.tv_usec - start.tv_usec) * 0.0001 + (end.tv_sec - start.tv_sec) * 1000;
    total_transmissionTime += transmissionTime;
    total_fileSize += fileSize;

    close(fd);
    close(dev_fd);
}
printf("Transmission time: %lf ms, File size: %ld bytes\n", total_transmissionTime, total_fileSize);
```

(2) device

① master device

當收到 user program 中 master 的通知後，從 memory 中找到要 mappping 的檔案位置，之後再藉由 ksocket 將資料傳給 slave device。

```
static int my_mmap(struct file* fp, struct vm_area_struct* vma);
void my_mmap_open(struct vm_area_struct* vma) {}
void my_mmap_close(struct vm_area_struct* vma) {}

//file operations
static struct file_operations master_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = master_ioctl,
    .open = master_open,
    .write = send_msg,
    .release = master_close,
    .mmap = my_mmap
};

struct vm_operations_struct mmap_vm_ops = {
    .open = my_mmap_open,
    .close = my_mmap_close
};
```

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

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

② slave device

一樣與 master device 類似，只是從發送端變接收端。當收到 ksocket 中從 master device 送過來的資料時後，從 memory 中找到要 output 的檔案

位置，之後再通知 user program 的 slave，由它負責 output 出檔案。

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

//file operations
static struct file_operations slave_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = slave_ioctl,
    .open = slave_open,
    .read = receive_msg,
    .release = slave_close,
    .mmap = my_mmap
};

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

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

```c
//reference https://linux-kernel-labs.github.io/refs/heads/master/labs/memory_mapping.html
static int my_mmap(struct file *filp, struct vm_area_struct *vma)
{
    vma->vm_pgoff = virt_to_phys((void *)filp->private_data)>>PAGE_SHIFT;
    int ret;
    ret = remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff, vma->vm_end-vma->vm_start, vma->vm_page_prot);
    if (ret < 0){
        pr_err("could not map the address area\n");
        return -EIO;
    }
    vma->vm_flags |= VM_RESERVED;
    vma->vm_private_data = filp->private_data;
    vma->vm_ops = &mmap_vm_ops;
    return 0;
}
```

(3) bonus

這部分我們直接改變 linux 內建 socket 資料結構中的 flags，如此一來就可以將 socket 由 synchronous 變成 asynchronous 模式。

```c
#ifdef BONUS
    sk->flags |= FASYNC;
#endif
```

socket 資料結構

```
struct socket
{
     socket_state  state;
     short type;
     unsigned long flags;
     struct socket_wq __rcu *wq;
     struct file *file;
     struct sock *sk;
     const struct proto_ops *ops;
}
```

(4) 說明自己程式的設計針對自己準備的測試檔設計的好處

mmap 在傳單獨一個很大的檔案會有優勢，但在傳很小的檔案時會因為 system overhead 反而可能比 fcntl 還沒有效率，所以我們設計一次傳數個很小的檔案應該可以突顯 mmap 在小檔案傳輸比較沒效率這件事。

我們自己設計了 4 組測資(分別對應到 our_input 的 4 個資料夾):

① 3 bytes 的小檔案 20 個，fcntl 應該比 mmap 有效率

② 520 bytes 的檔案 10 個，也就是介於 buffer size (512)和 map size (40960)之間的檔案，應該 mmap 比較有效率

③ 40959 bytes 的檔案 10 個，略小於 map size

④ 40961 bytes 的檔案 10 個，略大於 map size

其中① & ②正是前面所提及，fcntl 與 mmap 在傳大小檔案的差別。

而③ & ④的比較，則關注在 mmap 的部分，因為④略大於 map size，所以應該會花費較多 system overhead，導致只是多出兩個 bytes，就要花上明顯更多的時間。

2. 比較 file I/O 和 memory-mapped I/O 的結果與效能差異

(1) 一般情況(synchronous)

① our input 1(3 bytes 的小檔案 20 個)

	master	slave
	fcntl	fcntl
Transmission time	1.858500 ms	3.400100 ms
File size	60 bytes	60 bytes
	master	slave
	fcntl	mmap
Transmission time	0.368600 ms	2.313400 ms
File size	60 bytes	60 bytes
	master	slave
	mmap	fcntl

Transmission time	0.420600 ms	1.483600 ms
File size	60 bytes	60 bytes
	master	slave
	mmap	mmap
Transmission time	0.348100 ms	1.300200 ms
File size	60 bytes	60 bytes

② our input 2(520 bytes 的檔案 10 個，也就是介於 buffer size (512)和 map size (40960)之間的檔案)

	master	slave
	fcntl	fcntl
Transmission time	0.651400 ms	1.214100 ms
File size	5200 bytes	5200 bytes
	master	slave
	fcntl	mmap
Transmission time	0.096700 ms	0.545900 ms
File size	5200 bytes	5200 bytes
	master	slave
	mmap	fcntl
Transmission time	0.235600 ms	0.670200 ms
File size	5200 bytes	5200 bytes
	master	slave
	mmap	mmap
Transmission time	0.105600 ms	0.485900 ms
File size	5200 bytes	5200 bytes

③ our input 3(40959 bytes 的檔案 10 個，略小於 map size)

	master	slave
	fcntl	fcntl
Transmission time	3.535100 ms	3.916300 ms
File size	409590 bytes	409590 bytes
	master	slave
	fcntl	mmap
Transmission time	2.693200 ms	3.020900 ms
File size	409590 bytes	409590 bytes
	master	slave
	mmap	fcntl
Transmission time	0.179800 ms	1.040800 ms
File size	409590 bytes	409590 bytes

	master	slave
	mmap	mmap
Transmission time	0.205400 ms	0.961100 ms
File size	409590 bytes	409590 bytes

④ our input 4(40961 bytes 的檔案 10 個，略大於 map size)

	master	slave
	fcntl	fcntl
Transmission time	2.956800 ms	4.459500 ms
File size	409610 bytes	409610 bytes
	master	slave
	fcntl	mmap
Transmission time	2.637300 ms	3.186500 ms
File size	409610 bytes	409610 bytes
	master	slave
	mmap	fcntl
Transmission time	0.371000 ms	1.409800 ms
File size	409610 bytes	409610 bytes
	master	slave
	mmap	mmap
Transmission time	0.203000 ms	1.645100 ms
File size	409610 bytes	409610 bytes

⑤ sample input 1

	master	slave
	fcntl	fcntl
Transmission time	0.704200 ms	1.085100 ms
File size	24146 bytes	24146 bytes
	master	slave
	fcntl	mmap
Transmission time	0.256200 ms	0.949800 ms
File size	24146 bytes	24146 bytes
	master	slave
	mmap	fcntl
Transmission time	0.193500 ms	0.567000 ms
File size	24146 bytes	24146 bytes
	master	slave
	mmap	mmap
Transmission time	0.127000 ms	0.553900 ms

File size	24146 bytes	24146 bytes

⑥ sample input 2(一個超大檔案)

	master	slave
	fcntl	fcntl
Transmission time	15.892200 ms	16.809400 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	fcntl	mmap
Transmission time	955.969500 ms	956.000300 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	mmap	fcntl
Transmission time	8.107400 ms	9.090900 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	mmap	mmap
Transmission time	3.566100 ms	4.196000 ms
File size	12022885 bytes	12022885 bytes

比較:

與原先預測的一樣，在傳送小檔案的時候，fcntl 與 mmap 幾乎沒有太大差別，有時候 mmap 甚至比 fcntl 還慢一些，因為 mmap 需要一次 copy 整個 page；而當傳送到接近一個 page 大小的檔案時(我們設計的測資③ & ④)，mmap 的優勢就來了，快得跟鬼一樣；而超大檔案更不用說，mmap 完全海放 fcntl。

至於在 fcntl 與 mmap 交錯使用的情況(一邊用 fcntl，一邊用 mmap)，則像是混合版。如果在 mmap 快的情況，master 使用 mmap，slave 使用 fcntl，則 master 會比 master 使用 fcntl 的情況省時，slave 跟與其他 slave 使用 fcntl 的情況不會有明顯變化。

最後看到③ & ④的比較，與我們先前預測的相同，fcntl 沒有太明顯的差異，但 mmap 卻在④多費一些時間，因為 copy 一次 page 後仍然還有些資料沒傳送到，需要再一次，因此浪費了些時間，不過只差一點點，與我們預測會差很多有點落差。

(2) bonus(asynchronous)

① our input 1(3 bytes 的小檔案 20 個)

	master	slave
	fcntl	fcntl
Transmission time	2.158000 ms	3.104900 ms
File size	60 bytes	60 bytes

	master	slave
	fcntl	mmap
Transmission time	0.244900 ms	1.886700 ms
File size	60 bytes	60 bytes
	master	slave
	mmap	fcntl
Transmission time	0.315300 ms	1.255200 ms
File size	60 bytes	60 bytes
	master	slave
	mmap	mmap
Transmission time	0.355700 ms	1.647200 ms
File size	60 bytes	60 bytes

② our input 2(520 bytes 的檔案 10 個，也就是介於 buffer size (512)和 map size (40960)之間的檔案)

	master	slave
	fcntl	fcntl
Transmission time	0.606600 ms	1.249300 ms
File size	5200 bytes	5200 bytes
	master	slave
	fcntl	mmap
Transmission time	0.355000 ms	0.635900 ms
File size	5200 bytes	5200 bytes
	master	slave
	mmap	fcntl
Transmission time	0.192100 ms	0.732900 ms
File size	5200 bytes	5200 bytes
	master	slave
	mmap	mmap
Transmission time	0.115900 ms	0.774600 ms
File size	5200 bytes	5200 bytes

③ our input 3(40959 bytes 的檔案 10 個，略小於 map size)

	master	slave
	fcntl	fcntl
Transmission time	905.163000 ms	905.426600 ms
File size	409590 bytes	409590 bytes
	master	slave
	fcntl	mmap

Transmission time	0.620800 ms	4.351200 ms
File size	409590 bytes	409590 bytes
	master	slave
	mmap	fcntl
Transmission time	0.183700 ms	1.263800 ms
File size	409590 bytes	409590 bytes
	master	slave
	mmap	mmap
Transmission time	0.168800 ms	0.977000 ms
File size	409590 bytes	409590 bytes

④ our input 4(40961 bytes 的檔案 10 個，略大於 map size)

	master	slave
	fcntl	fcntl
Transmission time	3.037400 ms	6.400500 ms
File size	409610 bytes	409610 bytes
	master	slave
	fcntl	mmap
Transmission time	0.724000 ms	2.881200 ms
File size	409610 bytes	409610 bytes
	master	slave
	mmap	fcntl
Transmission time	0.335500 ms	1.786000 ms
File size	409610 bytes	409610 bytes
	master	slave
	mmap	mmap
Transmission time	0.312800 ms	1.519100 ms
File size	409610 bytes	409610 bytes

⑤ sample input 1

	master	slave
	fcntl	fcntl
Transmission time	0.685700 ms	1.369800 ms
File size	24146 bytes	24146 bytes
	master	slave
	fcntl	mmap
Transmission time	0.307900 ms	0.826300 ms
File size	24146 bytes	24146 bytes
	master	slave

	mmap	fcntl
Transmission time	0.164900 ms	0.659300 ms
File size	24146 bytes	24146 bytes
	master	slave
	mmap	mmap
Transmission time	0.262500 ms	0.610100 ms
File size	24146 bytes	24146 bytes

⑥ sample input 2(一個超大檔案)

	master	slave
	fcntl	fcntl
Transmission time	994.619100 ms	994.906700 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	fcntl	mmap
Transmission time	991.867700 ms	991.896300 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	mmap	fcntl
Transmission time	14.075000 ms	14.819900 ms
File size	12022885 bytes	12022885 bytes
	master	slave
	mmap	mmap
Transmission time	5.191300 ms	6.334100 ms
File size	12022885 bytes	12022885 bytes

比較:

結果與 synchronous 情況大同小異，在此不再贅述。

(3) synchronous & asynchronous 的比較

透過觀察可以發現，asynchronous 整體上比 synchronous 稍慢了些(比較沒效率)，因為 asynchronous 的情況，CPU 並不會等待當前 I/O 完成，而會直接先往下處理後面的事，直到 I/O 完成才會來處理，而此時時間仍然持續計時，所以導致 asynchronous 稍慢。

(4) page descriptors

我們有輸出在 output 資料夾中，於此不再重貼，煩請助教至 output 資料夾內查閱。

3.組內分工表

系級	學號	姓名	工作	分工比重

資工二	B07902096	李佳恩	slave 程式實作、開 repo	16.67%
資工二	B07902144	彭約博	master 程式實作	16.67%
數學三	B06201018	張芷榕	slave device 程式實作、bonus、生測資	16.67%
資工三	B06902131	吳冠穎	master device 程式實作	16.67%
資工三	B06902075	林詩敏	實驗、demo	16.67%
資工三	B06902136	賴冠毓	撰寫 report、統整並分析結果	16.67%

我們平均分工，所以每人分工比重皆為$\frac{1}{6} \approx 16.67\%$

4. reference

(1)https://github.com/wangyenjen/OS-Project-2
(2)https://github.com/qazwsxedcrfvtg14/OS-Proj2
(3)https://linux-kernel-labs.github.io/refs/heads/master/labs/memory_mapping.html