2020 OS Project 2
Synchronous Virtual Device Report

B07902058 資工二 楊盛評
B07902062 資工二 張彧瑋
B07902066 資工二 黃禹喆
B07902068 資工二 陳柏豪
B07902080 資工二 黃義峰

Programming design

Device

我們在 master_device 和 slave_device 中增加 mmap 的部分。
Device 的作法是將要傳送或接收 socket 的資料先放在自己檔案的 private_data 裡，再做 socket I/O，而這些資料我們採用會被複寫的模式。

master_device.c 和 slave_device.c 的共同 mmap 部分程式碼。其中 slave device 中下列部分除了 function 命名由 master 改為 slave 其餘相同。

```
static int master_map(struct file *file, struct vm_area_struct *vma);
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;
}

static struct file_operations master_fops = {
      .owner = THIS_MODULE,
      .unlocked_ioctl = master_ioctl,
      .open = master_open,
      .write = send_msg,
      .release = master_close,
      .mmap = master_map
};
```

```
static int master_map(struct file *file, struct vm_area_struct *vma){
       unsigned long start_addr = (virt_to_phys(file->private_data) >> 12);
       unsigned long size = vma->vm_end - vma->vm_start;
       if(remap_pfn_range(vma,vma->vm_start, start_addr, size, vma-
>vm_page_prot)){
              printk("remap fail !\n");
              return EAGAIN;
       }
    vma->vm_ops = &master_remap_vm_ops;
    vma->vm_flags |= VM_RESERVED;
    vma->vm_private_data = file->private_data;
     master_vma_open(vma);
    return 0;
}
```

master_device.c 中 mmap 程式碼

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

static ssize_t send_msg(struct file *file, const char __user *buf,
size_t count, loff_t *data){
       char msg[BUF_SIZE];
       if(copy_from_user(msg, buf, count))
              return -ENOMEM;
       ksend(sockfd_cli, msg, count, 0);
       return count;
}
```

slave_device.c 中 mmap 程式碼

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

ssize_t receive_msg(struct file *filp, char *buf, size_t count, loff_t
*offp ){
       char msg[BUF_SIZE];
       size_t len;
       len = krecv(sockfd_cli, msg, sizeof(msg), 0);
       if(copy_to_user(buf, msg, len))
              return -ENOMEM;
       return len;
}
```

Master

Master program 中, 先使用 mmap 映射到 master_device:

kernel_address = mmap(NULL, MAP_SIZE, PROT_WRITE, MAP_SHARED, dev_fd, 0);

然後使用 for 迴圈處理多個 input_file。每一次 for 迴圈中，打開當次處理的 input_file 並使用 mmap 映射。之後根據 method 的方法以 switch 選擇使用 fcntl 或是 mmap。

以下展示為 mmap 部分通過 while 將 file 寫入 device 的過程。remain 代表當前檔案還未處理的長度。以定好的 MAP_SIZE(32*page 大)對 input_file 進行切分處理。目前 remain 大於一個 MAP_SIZE 時，使用 memcpy 將一個 MAP_SIZE 的內容寫入 device 中，並修改 remain 和 offset。當 remain 小於一個 MAP_SIZE 大小時，則將此次寫入的 len 由 MAP_SIZE 改為 remain 進行寫入。

因為每次的 mmap 時 offset 一定要是 PAGE_SIZE 的倍數 ，因此一定要等到最後才補上不足一個 PAGE_SIZE 的資料，不然會造成錯誤

master.c 中 mmap 程式碼

```
#define PAGE_SIZE 4096
#define MAP_SIZE 4096*32
long long remain = (long long)file_size , offset = 0;

while( remain > 0){
      size_t len;
      if(remain > MAP_SIZE){
            len = MAP_SIZE;
            file_address = mmap(NULL, MAP_SIZE, PROT_READ, MAP_SHARED,
file_fd, offset);
            ioctl(dev_fd, 9020 , file_address);
            offset += len;
            remain -= len;
            memcpy(kernel_address, file_address , len);
            ioctl(dev_fd, 0x12345678, len);
      }
      else{
            len = remain;
            file_address = mmap(NULL, MAP_SIZE, PROT_READ, MAP_SHARED,
file_fd, offset);
            ioctl(dev_fd, 9020 , file_address);
            offset += len;
            remain -= len;
            memcpy(kernel_address, file_address , len);
            ioctl(dev_fd, 0x12345678, len);
      }
}
```

Slave

Slave 中同樣先對 slave_device 進行 mmap:
kernel_address = mmap(NULL, MAP_SIZE, PROT_WRITE, MAP_SHARED, dev_fd, 0);
然後用 for 迴圈處理需 output 的 file 數量。

以下展示的 mmap 部分中，通過 while 迴圈從 slave_device 中(使用 mmap)不斷接收 master 端傳來的 data(同一個 file 因封包而進行了多次的傳輸)。其中 offset 代表的是目前已經寫入 output_file 中的位置，ret 代表當次接收到的長度。將 ret 長度的內容 memcpy 至 output_file 中，修改 offset 並重複至當前 file 傳輸完畢(ret == 0)。再寫入之前先用 ftruncate 增加檔案大小，以防止 map 超出檔案大小所造成的 Bus Error，同 master，再寫入時也是以一個 PAGE_SIZE 為單位以防止 MMAP offset 所造成的錯誤。但在從 device 讀取資料時，可能會有未滿一個 PAG_SIZE 的情況發生，因此我們多增加了一個 DATA_BUFFER 來儲存資料，等資料集滿一個 PAGE 時再寫入目標檔案，最後再處理剩下不足一個 PAGE 的資料。

slave.c 中 mmap 程式碼

```
case 'm':
      offset = 0;
      char buffer[PAGE_SIZE * 3];
      int str_size = 0;
      do{
            ret = ioctl(dev_fd , 0x12345678);
            ftruncate(file_fd , offset + ret);
            file_address = mmap(NULL, ret, PROT_WRITE, MAP_SHARED,
file_fd, offset);
            if(file_address == NULL)      printf("file err\n");
            if(kernel_address == NULL)    printf("kernel err\n");
            memcpy(file_address, kernel_address , ret);
            offset += ret;
            file_size += ret;
      }while( ret > 0);
      ioctl(dev_fd, 9020, kernel_address);
      break;
```

Comparison

我們對 fcntl-fcntl 與 mmap-mmap 的結果進行比較，並觀察隨著檔案大小變化兩者之間的效能差異：

	target_file_1	target_file_2	target_file_3	target_file_4	target_file_5
size(bytes)	32	859	1343	1663	2042
FF(ms)	0.138	0.3226	0.2422	0.269	0.2125
MM(ms)	0.1325	0.4058	0.3297	0.3498	0.2794
	target_file_6	target_file_7	target_file_8	target_file_9	target_file_10
size(bytes)	3065	2808	4056	3659	4619
FF(ms)	0.2098	0.3125	0.3331	0.3099	0.347
MM(ms)	0.3351	0.2824	0.2882	0.3002	0.2368
	sample_2_target_file	1K	32K	512K	1M
size(bytes)	12022885	1024	32768	524288	1048576
FF(ms)	13.1811	0.0742	0.1452	3.7033	6.4859
MM(ms)	5.6989	0.1624	0.3295	0.5071	4.5001
	32M				
size(bytes)	33554432				
FF(ms)	51.0036				
MM(ms)	22.2943				

從上方圖表，我們可以觀察到，隨著檔案的大小上升，memory-mapping 的時間相較於 File I/O 少很多；反之，在檔案小時，File I/O 則有較好的效率。
這可能是因為雖然 mmap 可以減少 copy 的時間，但 mmap operation 的 overhead 大部分的時候較 copy 所的花費來的大，而這在小檔案時尤其明顯。這些 overhead 包括 page table 的建立、page fault 與 TLB miss 所造成的 memory access 時間的浪費等等。
然而在大檔案時，因為系統已經將部分檔案的記憶體位置紀錄在 page table 中，因此可大幅減少 Disk I/O 中的 seeking 的時間，降低 I/O phase 所產生的 latency。
另外，我們發現兩種方法在第一次執行時都有非常大的 overhead，這應該與 Disk I/O 所產生的 latency 有關。

Job responsibility:

B07902058 楊盛評：整體程式架構
B07902062 張彧瑋：Report, 程式測式與結果分析
B07902066 黃禹喆：Debug, 部分程式架構, demo
B07902068 陳柏豪：Debug, 部分程式架構
B07902070 黃義峰：Report, 程式架構整理

Reference:

https://www.cnblogs.com/pengdonglin137/p/8149859.html

https://www.itread01.com/content/1549440746.html

https://zhuanlan.zhihu.com/p/67053210

https://man7.org/linux/man-pages/man2/mmap.2.html

https://www.kernel.org/doc/gorman/html/understand/understand006.html

https://static.lwn.net/images/pdf/LDD3/ch03.pdf