OS-Project2

Programming design

sample code 提供 kernel 2.6 跟 kernel 4.14.25 兩個版本，為了能順利實作，我們先查找了 Ubuntu 的相應 kernel 版本，發現 Ubuntu 11.04 前的版本，kernel 都是 2.6.x，然後實際在跑的時候有碰到一些困難，再試了幾個版本後，我們最終選用 Ubuntu 12.04.x 順利把 sample code compile 成功。

此次 project 的目標在於實作 mmap，主要有四個部分的 code 需要實作，分別在：

- master.c
- master_device/master_device.c
- slave.c
- slave_device/slave_device.c

以下會分別做些簡單討論。

master.c | slave.c

在 master 須作以下操作：

1. 首先在 master.c 透過 mmap 來設定相對應的 file 及 kernel 的 address：`file_address`, `kernel_address`
2. 接著再利用 memcpy 將檔案從 `file_address` 複製到 `kernel_address`
3. 利用 munmap 解除 memory 的 mapping

```
// master.c
case 'm':
  while (offset < file_size) {
    size_t map_length = MAP_SIZE;
    if (file_size < map_length + offset) {
      map_length = file_size - offset;
    }

    // do mmap
    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, 0);

    // copy memory
    memcpy(kernel_address, file_address, map_length);

    offset += map_length;
    ioctl(dev_fd, master_IOCTL_MMAP, map_length);
```

```
    int cnt;
    for(cnt = 0; PAGE_SIZE * cnt < map_length; cnt++)
      ioctl(dev_fd, 0, file_address + PAGE_SIZE * cnt);

    // release mmap
    munmap(file_address, map_length);
    munmap(kernel_address, map_length);
  }
  break;
```

同樣的 slave 也做類似處理：

1. 首先在 slave.c 透過 mmap 來設定相對應的 file 及 kernel 的 address：`file_address`, `kernel_address`
2. 接著再利用 memcpy 將檔案從 `kernel_address` 複製到 `file_address`
3. 利用 munmap 解除 memory 的 mapping

```
// slave.c
case 'm':
  while (1) {
    ret = ioctl(dev_fd, master_IOCTL_MMAP);
    if (ret == 0) {
      file_size = offset;
      break;
    }

    // mmap
    posix_fallocate(file_fd, offset, ret);
    file_address = mmap(NULL, ret, PROT_WRITE, MAP_SHARED, file_fd, offset);
    kernel_address = mmap(NULL, ret, PROT_READ, MAP_SHARED, dev_fd, offset);


    // copy momery
    memcpy(file_address, kernel_address, ret);
    offset += ret;
    int cnt = 0;
    while(PAGE_SIZE * cnt < ret) {
      ioctl(dev_fd, 0, file_address + PAGE_SIZE * cnt);
      cnt++;
    }

    // release mmap
    munmap(file_address, ret);
    munmap(kernel_address, ret);
  }
  break;
```

master_device/master_device.c | slave_device/slave_device.c

接著要針對 master_IOCTL_MMAP/slave_IOCTL_MMAP 做設計。 在 master 的部分，我們用 ksocket 的 ksend 來傳送資料：

```
// master_device/master_device.c
case master_IOCTL_MMAP:
  ksend(sockfd_cli, file->private_data, ioctl_param, 0);
  break;
```

slave 則用 ksocket 的 krecv 來接收資料到 buffer 中，接著再用 memcopy 把檔案寫入 slave_device。

```
// slave_device/slave_device.c
case slave_IOCTL_MMAP:
  while(1){
    len = krecv(sockfd_cli, buf, sizeof(buf), 0);
    if (len == 0)
      break;
    memcpy(file->private_data+data_size, buf, len);
    data_size += len;
    if (data_size >= MAP_SIZE)
      break;
   }

  ret = data_size;
  data_size = 0;
  break;
```

The Result

實驗結果圖

file1

- kernel socket (TCP) with synchronous file I/O大概花 0.04ms
- memory-mapped I/O 也差不多 0.04ms

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 1 f
Transmission time: 0.055400 ms, File size: 4 bytes
Transmission time: 0.041100 ms, File size: 4 bytes
[ 2799.121510] connected to : 127.0.0.1 2325
[ 2799.122150] aceept sockfd_cli = 0xffff8800162d8280
[ 2799.122153] got connected from : 127.0.0.1 32776
[ 2799.631856] sockfd_cli = 0xffff8800162d8500  socket is created
[ 2799.631906] connected to : 127.0.0.1 2325
[ 2799.632739] aceept sockfd_cli = 0xffff8800162d8280
[ 2799.632742] got connected from : 127.0.0.1 32777
[ 2800.169833] sockfd_cli = 0xffff8800162d8500  socket is created
```

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 1 m
Transmission time: 0.053400 ms, File size: 4 bytes
Transmission time: 0.044400 ms, File size: 4 bytes
[ 2902.325325] master: 800000003FBC6025
[ 2902.325624] connected to : 127.0.0.1 2325
[ 2902.325649] slave: 800000002EABB867
[ 2902.808576] sockfd_cli = 0xffff8800162d8280  socket is created
[ 2902.809305] aceept sockfd_cli = 0xffff8800162d8500
[ 2902.809308] got connected from : 127.0.0.1 32799
[ 2902.809344] master: 800000003FBC6025
[ 2902.809584] connected to : 127.0.0.1 2325
```

file2

- kernel socket (TCP) with synchronous file I/O大概花 0.06ms
- memory-mapped I/O 則要花 0.17ms

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 2 f
Transmission time: 0.060200 ms, File size: 577 bytes
Transmission time: 0.047500 ms, File size: 577 bytes
[ 3120.554709] connected to : 127.0.0.1 2325
[ 3120.555238] aceept sockfd_cli = 0xffff8800162d8280
[ 3120.555240] got connected from : 127.0.0.1 32829
[ 3121.278970] sockfd_cli = 0xffff8800162d8500  socket is created
[ 3121.279021] connected to : 127.0.0.1 2325
[ 3121.279575] aceept sockfd_cli = 0xffff8800162d8280
[ 3121.279577] got connected from : 127.0.0.1 32830
[ 3122.153103] sockfd_cli = 0xffff8800162d8500  socket is created
```

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 2 m
Transmission time: 0.173200 ms, File size: 577 bytes
Transmission time: 0.158600 ms, File size: 577 bytes
[ 3441.246596] connected to : 127.0.0.1 2325
[ 3441.247127] aceept sockfd_cli = 0xffff8800162d8500
[ 3441.247129] got connected from : 127.0.0.1 32837
[ 3441.247160] master: 800000003F277025
[ 3441.247161] master: 80000000340E1025
[ 3441.247386] slave: 80000000212EB867
[ 3441.247388] slave: 8000000030D04867
[ 3452.781915] sockfd_cli = 0xffff8800162d8500  socket is created
```

file3

- kernel socket (TCP) with synchronous file I/O大概花 0.09ms
- memory-mapped I/O 則只要花 0.08ms

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 3 f
Transmission time: 0.080900 ms, File size: 9695 bytes
Transmission time: 0.095700 ms, File size: 9695 bytes
[ 3471.912176] aceept sockfd_cli = 0xffff8800162d8a00
[ 3471.912179] got connected from : 127.0.0.1 32839
[ 3471.912431] connected to : 127.0.0.1 2325
[ 3479.822750] sockfd_cli = 0xffff8800162d8280  socket is created
[ 3479.822793] connected to : 127.0.0.1 2325
[ 3479.823045] aceept sockfd_cli = 0xffff8800162d8a00
[ 3479.823048] got connected from : 127.0.0.1 32840
[ 3487.252327] sockfd_cli = 0xffff8800162d8280  socket is created
```

```
root@ubuntu:~/OS2019-project2/user_program# ./testrun.sh 3 m
Transmission time: 0.084500 ms, File size: 9695 bytes
Transmission time: 0.070400 ms, File size: 9695 bytes
[ 3663.410160] slave: 800000003E58D867
[ 3663.410161] slave: 800000001B2D9867
[ 3663.410162] slave: 800000003E58C867
[ 3663.410164] slave: 80000000328F2867
[ 3663.410165] slave: 800000003EAFE867
[ 3663.410166] slave: 80000000317BC867
[ 3663.410167] slave: 800000003E57F867
[ 3663.410169] slave: 800000002D29B867
```

file4

- kernel socket (TCP) with synchronous file I/O大概花 18ms
- memory-mapped I/O 則只要花 14ms。

```
root@fang-VirtualBox:/home/fang/Downloads/OS2019-project2-master/user_program# ./
testrun.sh 4 f
[ 3817.727999] slave: 80000000152DC867
[ 3817.728000] slave: 80000000152DD867
[ 3817.728002] slave: 80000000152DE867
[ 3817.728003] slave: 80000000152DF867
[ 3817.728005] slave: 80000000156D8867
[ 3817.728006] slave: 80000000156D9867
[ 3817.728008] slave: 80000000156DA867
[ 3817.728009] slave: 80000000156DB867
[ 3817.728010] slave: 8000000015034867
[ 3817.728012] slave: 8000000015035867
[ 3817.728013] slave: 8000000015036867
[ 3817.728015] slave: 8000000015037867
[ 3817.728016] slave: 8000000015698867
[ 3817.728018] slave: 8000000015699867
[ 3858.772055] sockfd_cli = 0xffff88001dbe4780  socket is created
root@fang-VirtualBox:/home/fang/Downloads/OS2019-project2-master/user_program# Ma
ster Transmission time: 14.611500 ms, File size: 1502860 bytes
Transmission time: 18.669700 ms, File size: 1502860 bytes
```

```
root@fang-VirtualBox:/home/fang/Downloads/OS2019-project2-master/user_program# ./
testrun.sh 4 m
Master Transmission time: 9.922500 ms, File size: 1502860 bytes
Transmission time: 14.537900 ms, File size: 1502860 bytes
[ 3650.695501] slave: 8000000000887A867
[ 3650.695502] slave: 8000000008879867
[ 3650.695504] slave: 8000000008878867
[ 3650.695505] slave: 8000000008877867
[ 3650.695507] slave: 8000000008876867
[ 3650.695508] slave: 8000000008891867
[ 3650.695509] slave: 8000000008890867
[ 3650.695511] slave: 800000000888F867
[ 3650.695512] slave: 800000000888E867
[ 3650.695514] slave: 800000000888D867
[ 3650.695515] slave: 800000000888C867
[ 3650.695517] slave: 800000000888B867
[ 3650.695518] slave: 800000000888A867
[ 3717.412797] e1000: eth0 NIC Link is Down
[ 3721.421083] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
root@fang-VirtualBox:/home/fang/Downloads/OS2019-project2-master/user_program#
```

Comparison & Discussion

1. file I/O 和 memory-mapped I/O 的 performance 差異

file I/O 需要把檔案內容寫進緩衝區內，建立 TCP 連綫對檔案內容進行傳輸。而 memory-map 則直接采用地址映射的方法進行檔案傳輸。所以檔案大小對 file I/O 的影響較大。

2. 兩個方法的穩定性差異

file I/O 傳輸檔案內容會受到網路環境以及 process scheduling 的影響。而 memory-map 的方法則較爲穩定。

3. 不同檔案大小的 performance 差異

可以觀察到檔案越大，需要的傳輸時間越長，然而在 file1~file3 增加的幅度並不大，推測是因為進行 file I/O 或 memory-map 時有 overhead，又 virtual machine 本身的效能較差，導致檔案大小對效能的影響沒有這麼顯著。另外檔案越大時， memory-mapped 更有優勢，因爲不用像 file I/O 需要進行檔案的搬移，而只需要對 memory 的映射表進行傳輸。因此， memory-mapped 的方法對於傳輸較大擋案時的 overhead 相對顯得越小。

4. 不同檔案大小的穩定性差異

file I/O 由於需要進行多次的檔案搬移，因此傳輸時間的對於檔案大小不同時會波動較大。而 memory-mapped I/O 的傳輸大小相對較爲固定，因此傳輸時間也較爲穩定。

Division of Labour

- 陳宣佑 D06944005：實作 mmap，Ubuntu 版本實驗
- 林懷宇 R06922032：debug，報告撰寫
- 陳　熙 R07922151：debug，分析結果
- 吳欣翰 D05944004：查找資料，測試資料傳輸
- 盧承億 B03902108：實作 mmap，Ubuntu 版本實驗
- 方淑玲 B06902064：查找資料，實作 mmap