2020 OS Project 2 - A/Synchronous Virtual Device

Group 56
資工三 b06902011 陳義誠
資工三 b06902019 洪佳生
資工三 b06902022 陳翰霆
資工三 b06902103 尹聖翔
資工三 b06902122 黃文彥
機械四 b05502060 沈允中

I. Programming Design

(1) Ksocket

- ksocket.c :
  Transmit data between *master device* and *slave device* and let the two devices can communicate through the socket. Files in this folder are the same as sample codes, nothing has been changed.

(2) User Program

- master.c :
  The main purpose of this program is transferring the given files to the master device interface using ioctl to specify our additional functions.
- slave.c :
  The input contains master IP, the main purpose of this program is to receive the files sent from master and write them into disk. It also uses ioctl to manipulate slave device driver's I/O-interface.

(3) Device Driver

- master_device.c :
  - This is the kernel module of master device, the spirit of this device. The roles of this program include receiving data from master.c, establishing ksocket and communicating with slave device.
- slave_device.c :
  - Similarly, this is the kernel module of slave device. The main difference is that it receives data from ksocket and forward the data o slave program.

(4) Implementation of mmap related functions

- User programs (master, slave)
  Deeming all resources as files, we first use mmap to map the file descriptor of disk file to user space virtual memory.
  Then, we use memcpy to copy the data.
- Device drivers (master_device, slave_device)
  First, we acquire a piece of space for mmap using kmalloc when we open the device. Use kfree when we close the device. This assures available space for files.

(5) Compile and Execute

Compile the source code

```
$sudo ./compile.sh
```

After this step, the modules of master device, slave device and ksocket will be generated. The environment should be settled if you use Linux 4.14.25 kernel.

Execute

To start the master program.

```
$cd user_program
$sudo ./master 1 file_in mmap
```

To start the slave program.

```
$cd user_program
$sudo ./slave 1 file_out fcntl 127.0.0.1
```

⋆ You can choose fcntl or mmap.

II. Experiments

(1) Testing Platform

- OS : Ubuntu 16.04 (virtualbox)
- Kernel : Linux kernel 4.14.25

(2) Result

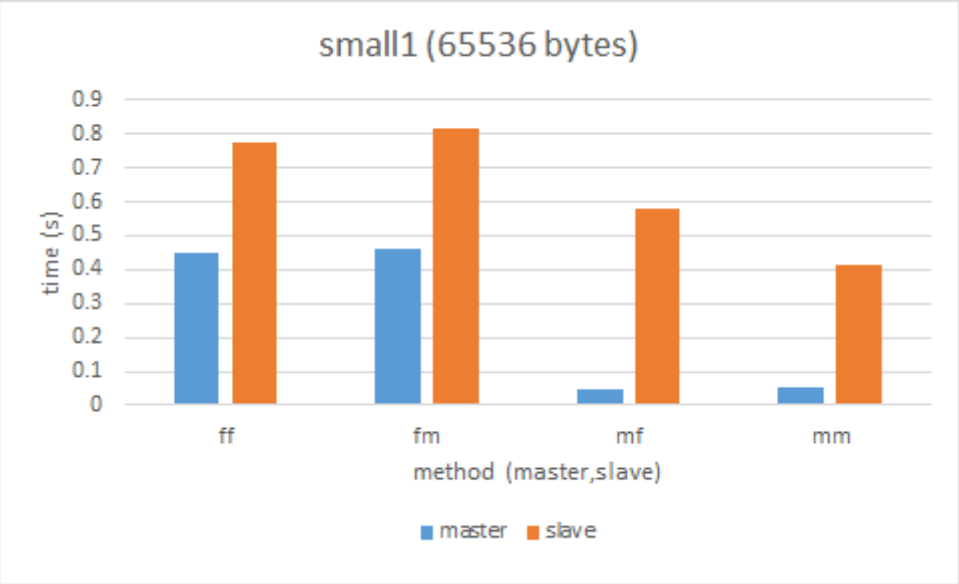
We implemented files transferring using memory mapping, and compare the performance with fcntl.
We use the transmission time as the benchmark.

ff : fcntl v.s. fcntl, **fm** : fcntl v.s. map
mf : mmap v.s. fcntl, **mm** : mmap v.s. mmap

- self define input

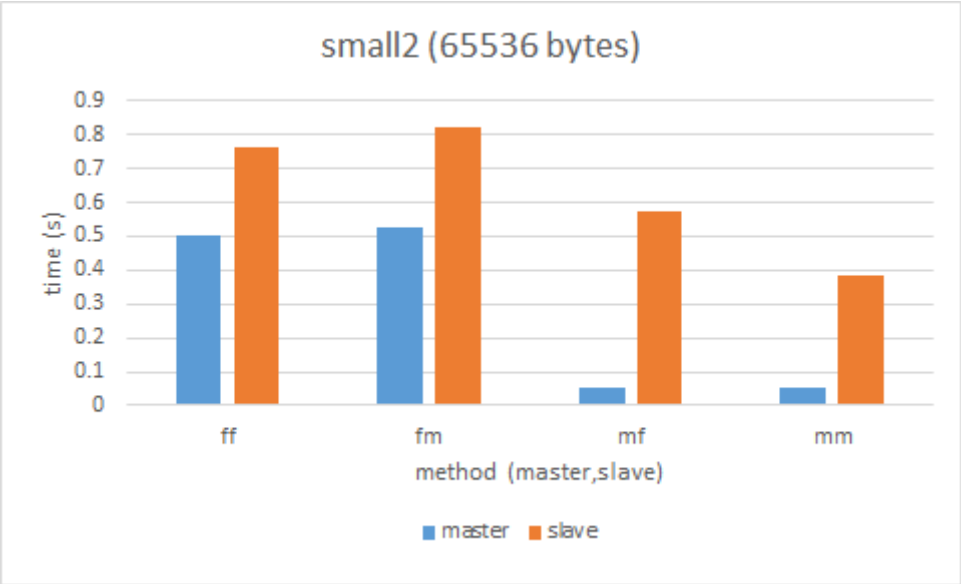
small1 (65536 bytes)

	ff	**fm**	**mf**	**mm**
master	0.447	0.459	0.047	0.053
slave	0.777	0.817	0.580	0.412



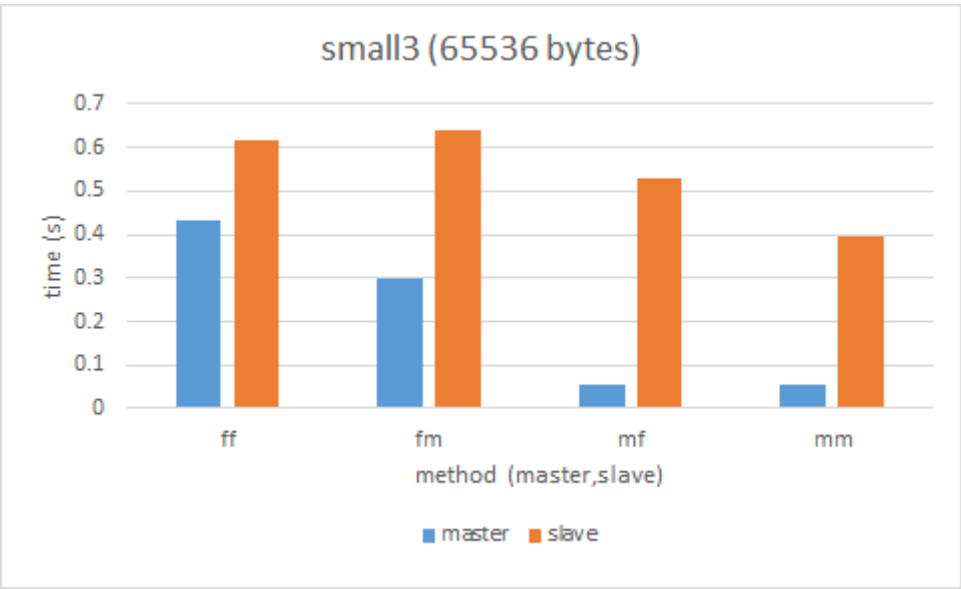
small2 (65536 bytes)

	ff	**fm**	**mf**	**mm**
master	0.500	0.525	0.053	0.051
slave	0.763	0.820	0.576	0.385



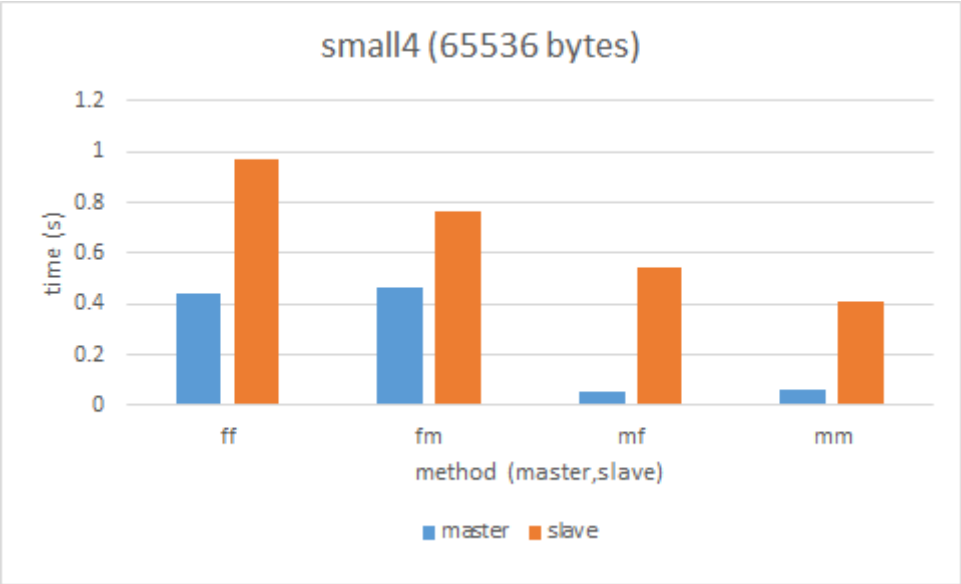
small3 (65536 bytes)

	ff	fm	mf	mm
master	0.434	0.301	0.056	0.054
slave	0.615	0.641	0.527	0.394



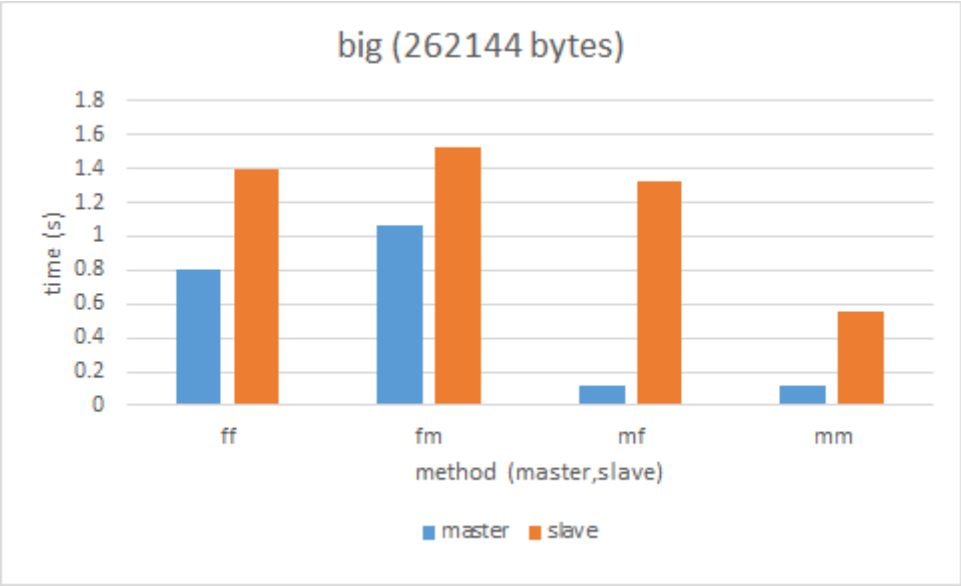
small4 (65536 bytes)

	ff	fm	mf	mm
master	0.443	0.461	0.057	0.063
slave	0.972	0.765	0.545	0.412



big (262144 bytes)

	ff	fm	mf	mm
master	0.809	1.068	0.123	0.122
slave	1.401	1.527	1.321	0.554



III. Analysis

- Result analysis
  When file size is small and execution time is less than 0.1ms, m/m doesn't appear good performance rather then f/f.
  However, as the increase of file size, f/f's transmission time increase dramatically.
- Compare Mmap I/O with File I/O
  When File I/O execute read and write, it need to call system call. In the other side, Mmap I/O can straightly operate local memory after call mmap().Thus, Mmap I/O can save time by avoiding calling system call and copying data to cache buffer.

It’s worth noting that File I/O may perform better while file size is small, because time of memory mapping is more than file transmitting.

- Results about our self define input
  For the large files, the file sizes are integer times of our page size. When we use memory mapping, this will me more efficient for file transferring.
  Comparing our two sets of input, transferring multiple small files and transferring a large file of the same size, we observed that transferring a single large file is more efficient. We think the reason is that there would be more overhead calling more system calls when transferring multiple small files. Therefore, by our observation, memory mapping is more favorable when transferring large files.

IV. Predicament

- When we implement the data transfer between master and slave,if master’s data upload speed is slower than slave’s data download speed,the program will crash because it will return 0 when receiving the data,so we use `MSG_WAITALL` flag in **`krecv`** to fix this problem.

V. Asynchronous

- We also implemented the asynchronous version in the ksocket. We just needed to slightly modify the source code to achieve the asynchronized version. In our experiment, Using synchronized method and asynchronized method don’t differ very much in speed. It speeded up about 15% using async version instead of sync version. We think the reason is that the bottleneck is not at this part, calling mmap or fcntl consumes much more time, so we didn’t see much difference using asynchronized version.

VI. Contribution

資工三 b06902011 陳義誠: Design master mmap features. Debug on slave mmap features.
資工三 b06902019 洪佳生: Design slave mmap features, writing report.
資工三 b06902022 陳翰霆: Dealing with linux kernel, writing report, experimenting
資工三 b06902103 尹聖翔: Design slave mmap features
資工三 b06902122 黃文彥: Write report, experiment and analysis.
機械四 b05502060 沈允中: Write report

VII. Reference

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

- ioctl
  https://www.man7.org/linux/man-pages/man2/ioctl.2.html (https://www.man7.org/linux/man-pages/man2/ioctl.2.html)

Supports from our classmates:
b06902012、b06902048 and wanyenjen/HigasaOR's github.