Homework Assignment #3:

Doing File I/O by Standard C Library, File I/O System Calls and Memory-Mapped Files

Outline

- **fopen()、fread()、fwrite()、fclose()**
- Homework Assignment #3_1
- open()、read()、write()、close()
- Homework Assignment #3_2
- Memory-mapped file
- Homework Assignment #3_3
- Homework Assignment #3_4

Standard C library

- **fopen()**

```
#include <stdio.h>
```

```
FILE *fopen ( const char *filename, const char *mode );
```

```
EX: FILE *f1;
      f1 = fopen ( “XXX.txt”, “w” );
```

mode:

“r”	**read:** Open file for input operations. The file must exist.
“w”	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
“a”	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. The file is created if it does not exist.

return value: Successful: the function returns a pointer to a ***FILE*** object. Otherwise: a null pointer is returned.

Standard C library(cont.)

- **fread()**

```
#include <stdio.h>
```

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
```

```
EX: fread ( buffer, sizeof(char), 4, f1 );
```

ptr: Pointer to a block of memory with a size of at least (***size*******count***) bytes.

size: Size, in bytes, of each element to be read. ***size_t*** is an unsigned integral type.

count: Number of elements, each one with a size of ***size*** bytes.

stream: Pointer to a ***FILE*** object that specifies an input stream.

return value: The total number of elements successfully read is returned. If this number differs from the count parameter, the proper indicator is set, which can be checked with ***ferror*** and ***feof.***

Standard C library(cont.)

- **fwrite()**

```
#include <stdio.h>
```

```
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
```

```
EX: fwrite ( buffer, sizeof(char), 4, f1 );
```

ptr: Pointer to the array of elements to be written, converted to a const void*.

size: Size, in bytes, of each element to be written. ***size_t*** is an unsigned integral type.

count: Number of elements, each one with a size of ***size*** bytes.

stream: Pointer to a ***FILE*** object that specifies an output stream.

return value: The total number of elements successfully written is returned. If this number differs from the count parameter, a writing error prevented the function from completing. In this case, the error indicator (***ferror***) will be set for the stream.

Standard C library(cont.)

- **fclose()**

```
#include <stdio.h>
```

```
int fclose ( FILE * stream );
```

```
EX: fclose (f1);
```

stream: Pointer to a ***FILE*** object that specifies the stream to be closed.

return value: If the stream is successfully closed, a zero value is returned. On failure, EOF is returned.

Outline

- fopen()、fread()、fwrite()、fclose()
- **Homework Assignment #3_1**
- open()、read()、write()、close()
- Homework Assignment #3_2
- Memory-mapped file
- Homework Assignment #3_3
- Homework Assignment #3_4

Homework Assignment #3_1

- Write two programs that use standard C library to copy one file, whose file size is 100 MB, to another file with different I/O size, and measure the file copy times.
- One program copies the file by issuing $25*2^{20}$ sequential 4B-sized I/O, and the other copies the file by issuing $25*2^{10}$ sequential 4KB-sized I/O.

- First step: Open the files using fopen().

- Second step: Copy files by issuing $25*2^{20}$ / $25*2^{10}$ sequential 4B-sized / 4KB-sized I/O using fread() and fwrite(). Besides, measure the execution time in this step.

- Third step: Close the files using fclose().

- Notably, clear the page cache before running your program.

Homework Assignment #3_1(cont.)

- gettimeofday()
  - Get the time as well as a timezone.

```
#include <sys/time.h>
```

```
int gettimeofday (struct timeval *tv, struct timezone *tz );
```

```
struct timeval {
        time_t tv_sec;          /* seconds */
        suseconds_t tv_usec;    /* microseconds */
};

struct timezone {
        int tz_minuteswest;     /* minutes west of Greenwich */
        int tz_dsttime;         /* type of DST correction */
};
```

Homework Assignment #3_1(cont.)

- EX:

```
int main()
{
        struct  timeval start;
        struct  timeval end;

        unsigned  long diff;
        gettimeofday(&start,NULL);
        delay(10);
        gettimeofday(&end,NULL);
        diff = 1000000 * (end.tv_sec-start.tv_sec)+ end.tv_usec-start.tv_usec;
        printf("the difference is %ld (us)\n”, diff);
        return 0;
}
```

Homework Assignment #3_1(cont.)

- If you want to take a look at the total available as well as used memory in the system. You can use **free** command.
- To clean the Linux buffer/cache, you can use **echo 3 > /proc/sys/vm/drop_caches** command. Notice that, you should run it in **root** mode.

- Ex:

```
root@oslab-ASUSPRO-D840MB-M840MB:/home/oslab# free
              total        used        free      shared  buff/cache   available
Mem:       16195052      836396    14732860      136168      625796    14946580
Swap:       2097148       84224     2012924
root@oslab-ASUSPRO-D840MB-M840MB:/home/oslab# echo 3 > /proc/sys/vm/drop_caches
root@oslab-ASUSPRO-D840MB-M840MB:/home/oslab#
```

Outline

- fopen()、fread()、fwrite()、fclose()
- Homework Assignment #3_1
- **open()、read()、write()、close()**
- Homework Assignment #3_2
- Memory-mapped file
- Homework Assignment #3_3
- Homework Assignment #3_4

Linux system call interface

- **open()**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open ( const char *filename, int flags );
Int open ( const char *filename, int flags, mode_t mode );
```

```
EX: f1 = open ( "hello.txt", O_WRONLY | O_TRUNC );
```

flags:

O_RDONLY	Read only.
O_WRONLY	Write only.
O_RDWR	Read and write.
O_CREAT	Create file if it doesn't exist.
O_TRUNC	If the file already exists and the open mode allows writing (i.e., is O_RDWR or O_WRONLY) it will be truncated to length 0.

return value: Return the file descriptor, or -1 is returned and ***errno*** is set appropriately.

Linux system call interface(cont.)

- **read()**

```
#include<unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count);
```

```
EX: read(f1, buffer, 4);
```

fd: The file descriptor of where to read the input.

buf: A character array where the read content will be stored.

count: The number of bytes to read before truncating the data. If the data to be read is smaller than nbytes, all data is saved in the buffer.

return value: Upon successful completion, it shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return -1 and set ***errno*** to indicate the error.

Linux system call interface(cont.)

- **write()**

```
#include<unistd.h>
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

```
EX: write(f1, buffer, 4);
```

fd: The file descriptor of where to write the input.

buf: A pointer to a buffer of at least nbytes bytes, which will be written to the file.

count: The number of bytes to write. If smaller than the provided buffer, the output is truncated.

return value: On success, the number of bytes written is returned. On error, -1 is returned, and ***errno*** is set to indicate the cause of the error.

Linux system call interface(cont.)

- **close()**

```
#include<unistd.h>
```

```
int close(int fd);
```

```
EX: close(f1);
```

fd: The file descriptor to be closed.

return value: returns zero on success. On error, -1 is returned, and **errno** is set appropriately.

Outline

- fopen()、fread()、fwrite()、fclose()
- Homework Assignment #3_1
- open()、read()、write()、close()
- **Homework Assignment #3_2**
- Memory-mapped file
- Homework Assignment #3_3
- Homework Assignment #3_4

Homework Assignment #3_2

- Write two programs that use Linux system call interface to copy one file, whose file size is 100 MB, to another file with different I/O size, and then measure the file copy time.
- One program copies the file by issuing $25*2^{20}$ sequential 4B-sized I/O, and the other copies the file by issuing $25*2^{10}$ sequential 4KB-sized I/O.

- First step: Open the files using open().

- Second step: Copy files by issuing $25*2^{20}$ / $25*2^{10}$ sequential 4B-sized / 4KB-sized I/O using read() and write(). Besides, measure the execution time in this step.

- Third step: Close the files using close().

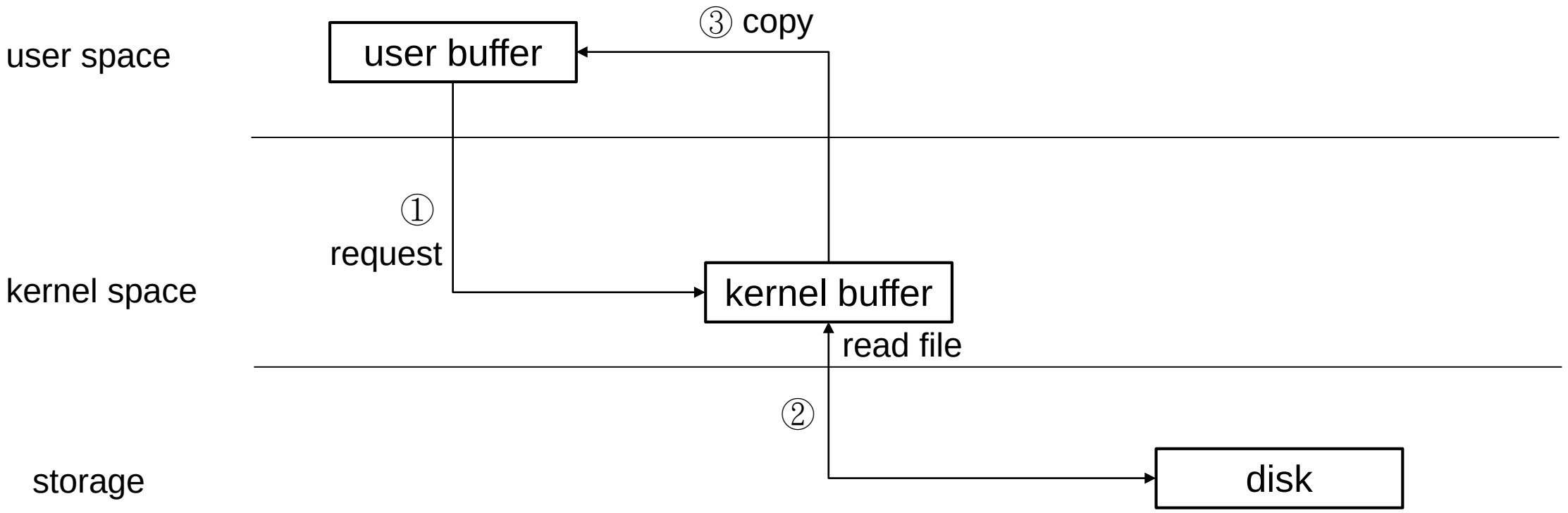
- Notably, clear the page cache before running your program.

Outline

- fopen()、fread()、fwrite()、fclose()
- Homework Assignment #3_1
- open()、read()、write()、close()
- Homework Assignment #3_2
- **Memory-mapped file**
- Homework Assignment #3_3
- Homework Assignment #3_4

Memory-mapped file

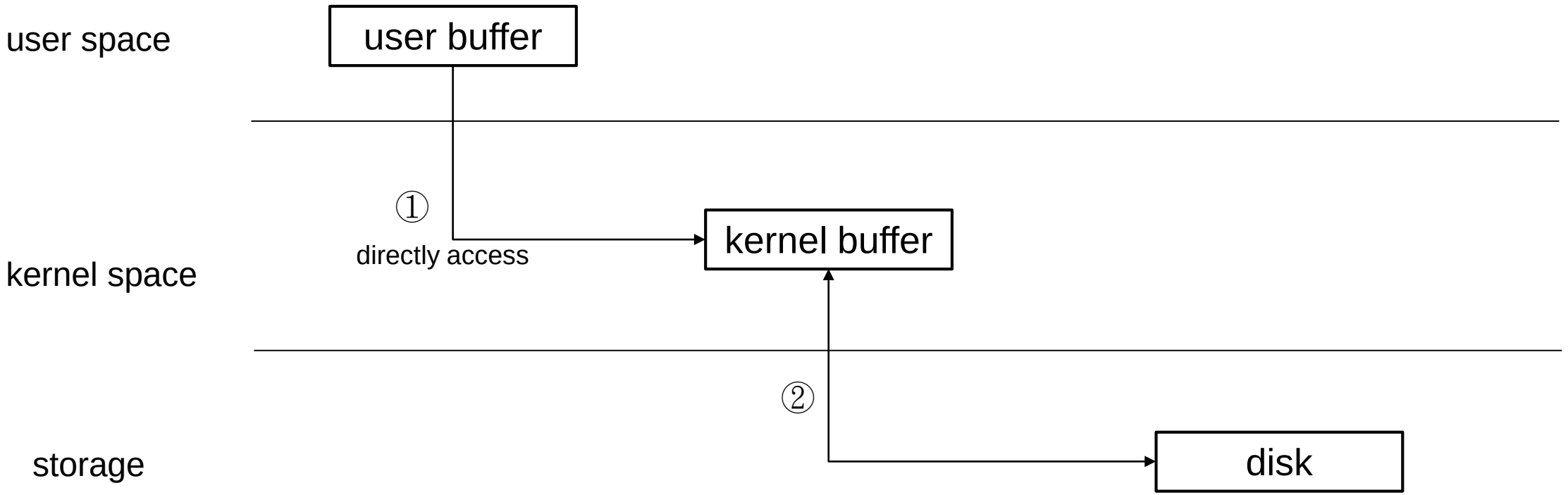
- Consider a sequential read of a file on disk using the standard system calls **open()**, **read()**, and **write()**. Each file access requires a system call and disk access.



Read system call

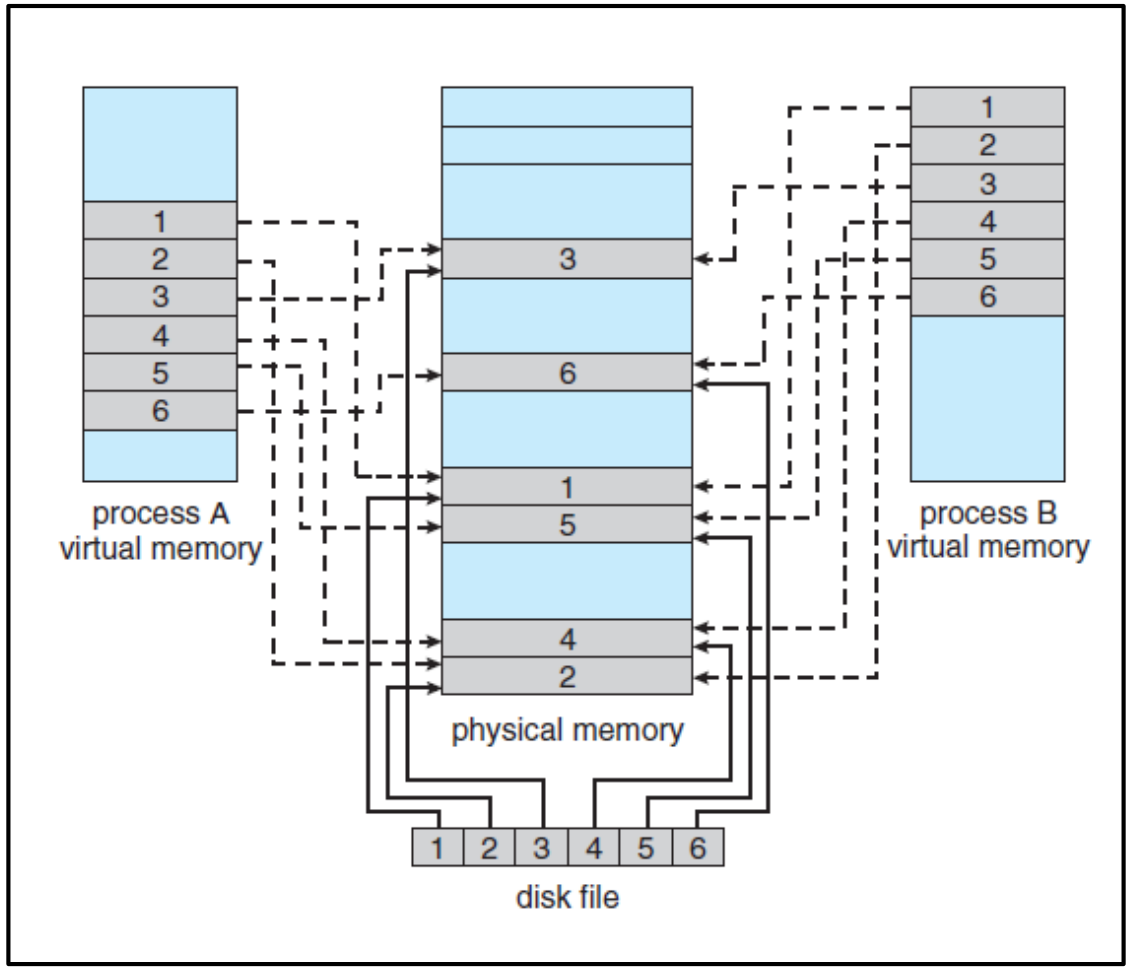
Memory-mapped file(cont.)

- We can use the virtual memory techniques to treat file I/O as routine memory accesses. This approach, known as **memory mapping** a file, allows a part of the virtual address space to be logically associated with the file.



memory-mapped read

Memory-mapped file(cont.)



Memory-mapped file(cont.)

- **mmap()**

```
#include <sys/mman.h>
```

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

```
EX: char *map_f1;
      map_f1 = mmap(NULL, filesize, PROT_READ, MAP_SHARED, f1, 0);
```

addr、length、fd、offset: The pages starting at **addr** and continuing for at most **length** bytes to be mapped from the object described by **fd**, starting at byte offset **offset**.

prot:

PROT_EXEC	Pages may be executed.
PROT_READ	Pages may be read.
PROT_WRITE	Pages may be written.
PROT_NONE	Pages may not be accessed.

Memory-mapped file(cont.)

- **mmap()**

flags:

MAP_SHARED	Share this mapping with all other processes that map this object. Storing to the region is equivalent to writing to the file. The file may not actually be updated until **msync**(2) or **munmap**(2) are called.
MAP_PRIVATE	Create a private copy-on-write mapping. Stores to the region do not affect the original file. It is unspecified whether changes made to the file after the **mmap**() call are visible in the mapped region.

return value: On success, ***mmap()*** returns a pointer to the mapped area. On error, the value **MAP_FAILED** (that is, (void *) -1) is returned, and **errno** is set appropriately.

Memory-mapped file(cont.)

- **munmap()**

```
#include <sys/mman.h>
```

```
int munmap(void *addr, size_t length);
```

```
EX: munmap(map_f1, filesize);
```

addr、length: The function remove any mappings for those entire pages containing any part of the address space of the process starting at **addr** and continuing for **len** bytes.

return value: Upon successful completion, it shall return 0; otherwise, it shall return -1 and set **errno** to indicate the error.

Outline

- fopen()、fread()、fwrite()、fclose()
- Homework Assignment #3_1
- open()、read()、write()、close()
- Homework Assignment #3_2
- Memory-mapped file
- **Homework Assignment #3_3**
- Homework Assignment #3_4

Homework Assignment #3_3

- Write two programs that use memory-mapped file to copy one file, whose file size is 100 MB, to another file, and then measure the file copy time.
- One program copies the file by issuing $25*2^{20}$ sequential 4B-sized I/O, and the other copies the file by issuing $25*2^{10}$ sequential 4KB-sized I/O.
- First step: Open the files using open().
- Second step: Create a new mapping in the virtual address space using mmap().
- Third step: Copy files by issuing $25*2^{20}$ / $25*2^{10}$ sequential 4B-sized / 4KB-sized I/O. Besides, measure the execution time in this step.
- Fourth step: Delete the mappings using munmap().
- Fifth step: Close the files using close().

Homework Assignment #3_3 (cont.)

- **truncate()**

```
#include<unistd.h>
```

```
int truncate(const char * path, off_t length);
```

```
EX: truncate(“output.txt", filesize);
```

path: a file name or path.

length: If the file previously was larger than **length**, the extra data is lost. If the file previously was shorter, it is extended, and the extended part reads as null bytes ('\0').

return value: returns zero on success. On error, -1 is returned, and **errno** is set appropriately.

Outline

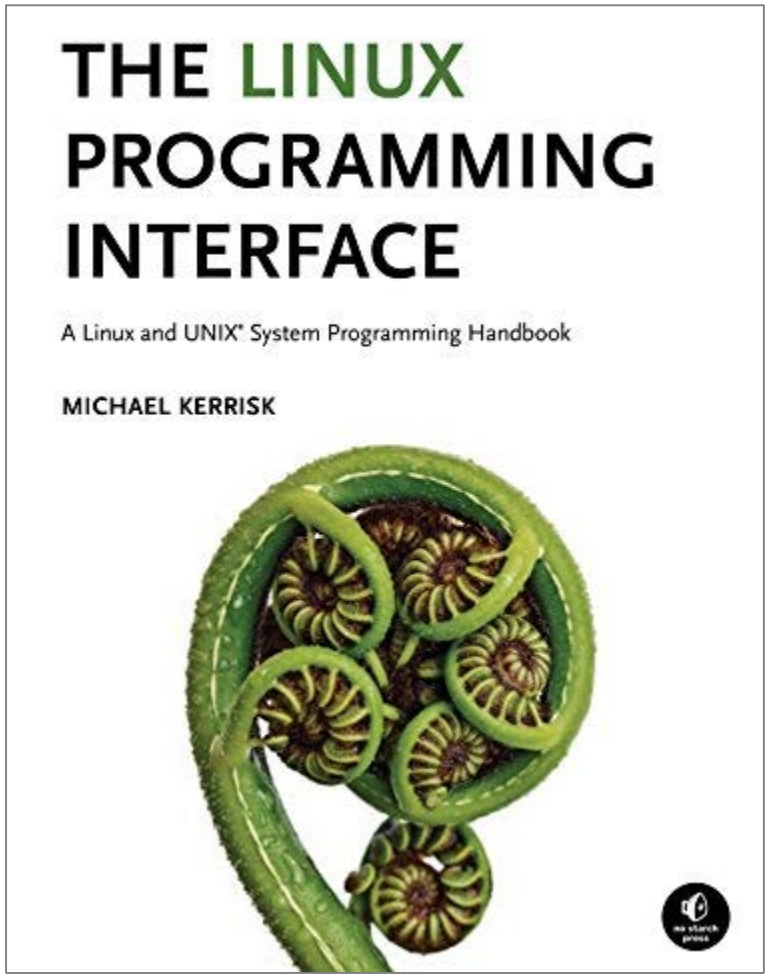
- fopen()、fread()、fwrite()、fclose()
- Homework Assignment #3_1
- open()、read()、write()、close()
- Homework Assignment #3_2
- Memory-mapped file
- Homework Assignment #3_3
- **Homework Assignment #3_4**

Homework Assignment #3_4

- Compare the times measured in Homework 3_1, 3_2, and 3_3 and explain the results.
- Notably,
  - Clear the page cache after running your program each time.

Reference

- The Linux Programming Interface: A Linux and UNIX System Programming Handbook.



Turn in

- Deadline
  2020/5/22 23:59:59
- Upload to i-learning
- File name
  - HW3_ID.zip (e.g. HW3_7105056035.zip)
    - Source code
    - Word file
- If you want to update
  - HW3_ID_New1.zip, HW3_ID_New2.zip….etc
- If you don’t hand in your homework on time, your score will be deducted 10 points every day.

TA

- Name: 蔡秉謙
- Email: g108056038@mail.nchu.edu.tw
  - Title format: HW3 - [your name]
- Lab: OSNET(1003A)