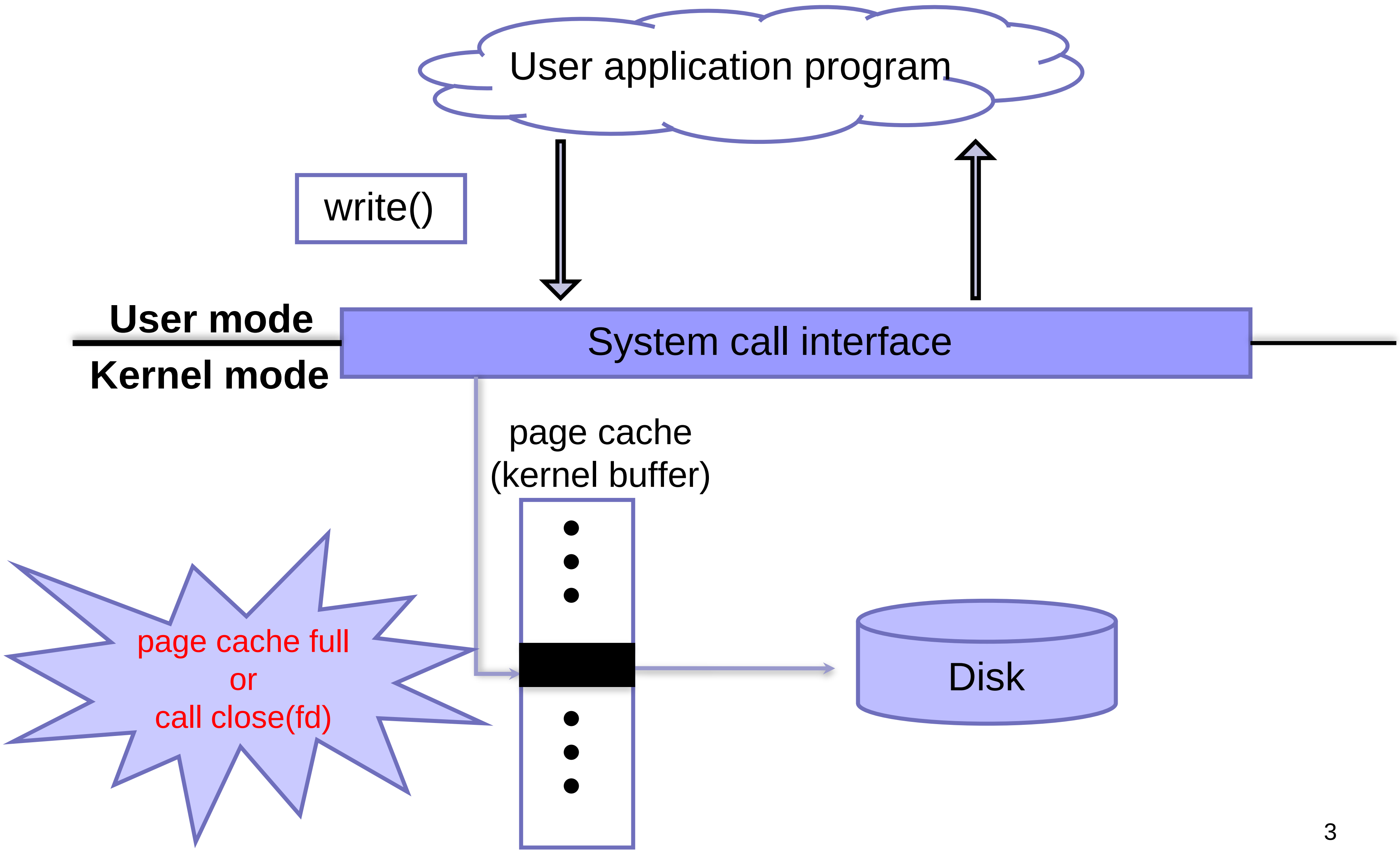
Homework Assignment #2

Regular file I/O by the system calls and C library layer APIs

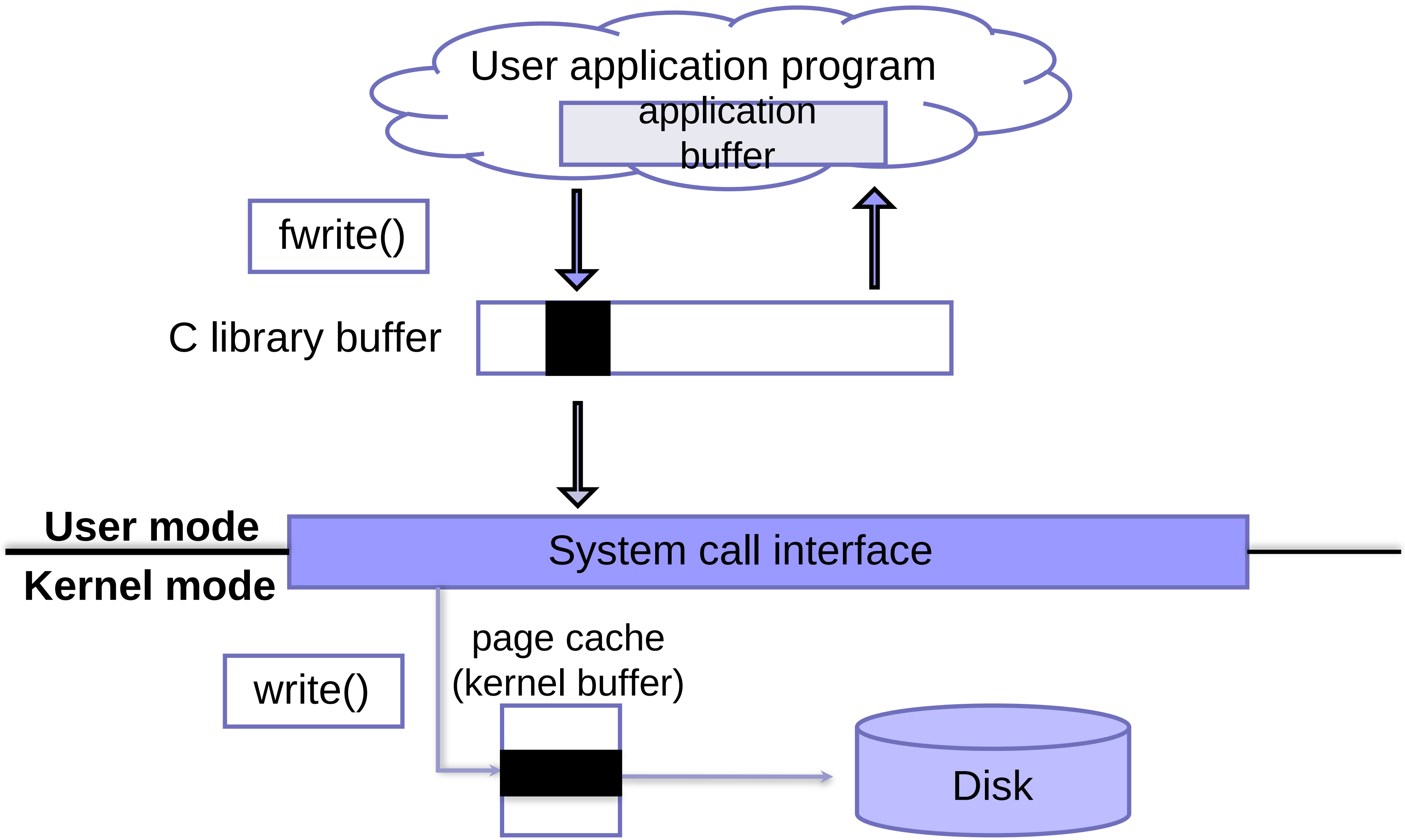
Outline

- Regular file I/O by the system calls
  - Kernel buffer (page cache)
  - I/O system calls
- Regular file I/O by the C library APIs
  - Application Buffer, C Library Buffer and Kernel Buffer
  - C library I/O APIs
- Time function
- Homework Assignment #2
- Reference

Kernel Buffer



Application Buffer, C Library Buffer and Kernel Buffer



The I/O System Calls: open()

- **open()**
  - Used to Open the file for reading, writing or both.

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int open (const char* Path, int flags [, int mode ]);
```

```
EX: int fd = open("myfile.txt", O_CREAT | O_WRONLY, 0600);
```

Path : path to file which you want to use.

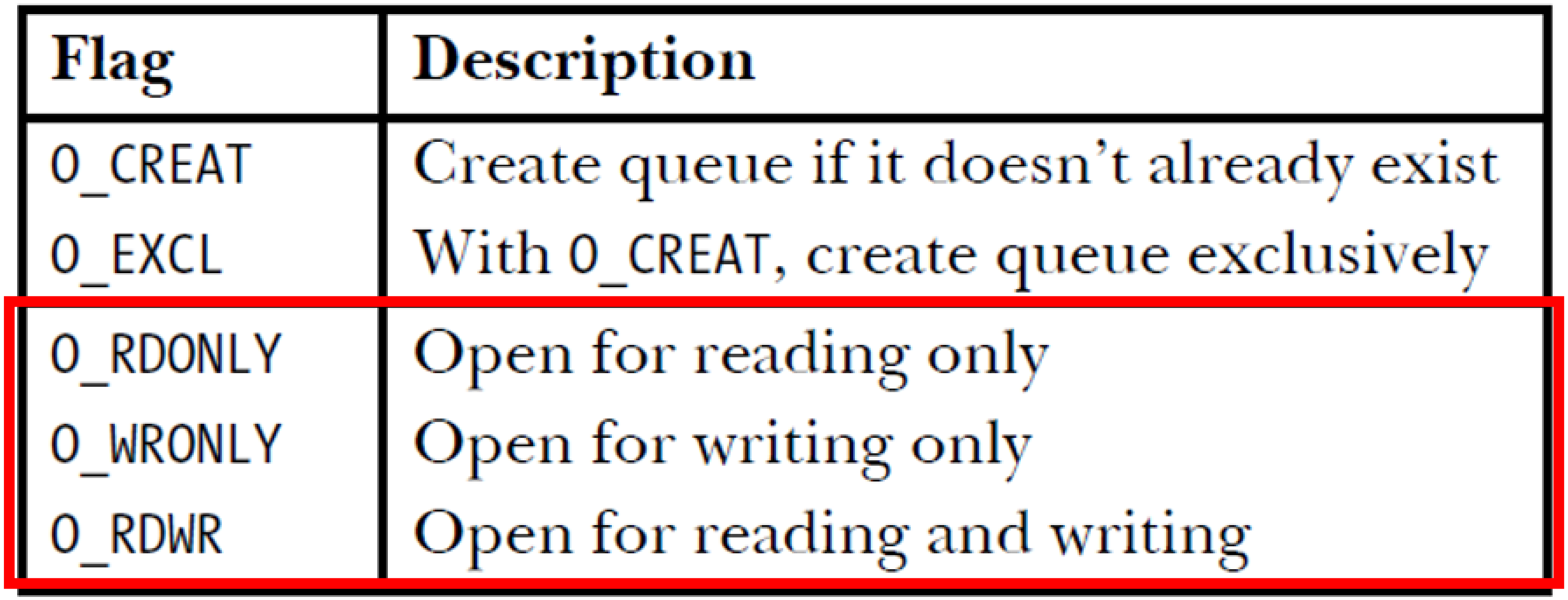
flags : How you like to use. (shown in next slide)

mode : A permission for new file.
(shown in the following next slide)

Return file descriptor(fd) used, -1 upon failure.

The I/O System Calls: open() (Cont.)

- **Bit values for the open()**
  - flags argument



Flag	Description
O_CREAT	Create queue if it doesn't already exist
O_EXCL	With O_CREAT, create queue exclusively
O_RDONLY	Open for reading only
O_WRONLY	Open for writing only
O_RDWR	Open for reading and writing

The I/O System Calls: open() (Cont.)

- **Bit values for the open()**
  - mode argument

Constant	Octal value	Permission bit
S_ISUID	04000	Set-user-ID
S_ISGID	02000	Set-group-ID
S_ISVTX	01000	Sticky
S_IRUSR	0400	User-read
S_IWUSR	0200	User-write
S_IXUSR	0100	User-execute
S_IRGRP	040	Group-read
S_IWGRP	020	Group-write
S_IXGRP	010	Group-execute
S_IROTH	04	Other-read
S_IWOTH	02	Other-write
S_IXOTH	01	Other-execute

The I/O System Calls: write()

- **write()**
  - Writes *cnt* bytes from *buf* to the file or socket associated with *fd*.

```
#include <fcntl.h>
size_t write (int fd, void* buf, size_t cnt);
```

```
EX: int size = write(fd, "e", strlen("e") );
```

fd : file descripter.

buf : buffer to write the data.

cnt : length of buffer.

If *buf* size less than the *cnt* then *buf* will lead to the overflow condition.
If *cnt* is zero, write() returns 0 without attempting any other action.

Return number of bytes written on success , -1 on error.

The I/O System Calls: close()

- **close()**
  - Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

```
#include <fcntl.h>
int close(int fd);
```

```
EX: close(fd);
```

fd : file descriptor

Return 0 on success, -1 on error.

Example

```
#include<stdio.h>  #include<sys/types.h>  #include<sys/stat.h>
#include<fcntl.h>   #include<unistd.h>
#include<stdlib.h>              //for exit()
#include <string.h>             //for strlen()

int main(){
            int fd;
            fd = open("myfile.txt", O_CREAT | O_WRONLY, 0600);
            if (fd < 0){
                        printf("Failed to open the file.\n");exit(1);       }
            int size;
            size = write(fd, "e", strlen("e") );
            close(fd);

            printf("length of write data=%d \n", strlen("e"));
            printf("Number of bytes written on success=%d \n\n", size);

            return 0;
}
```

Outline

- **Regular file I/O by the system calls**
  - Kernel buffer (page cache)
  - I/O system calls
- **Regular file I/O by the C library APIs**
  - Application Buffer, C Library Buffer and Kernel Buffer
  - C library I/O APIs
- **Time function**
- **Homework Assignment #2**
- **Reference**

The C library I/O APIs: fopen()

- **fopen**
  - Opens the filename pointed to , and uses the given mode.

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

```
EX: FILE *fp;
      fp = fopen("myfile.txt", "wt+");
```

filename : This is the C string containing the name of the file
to be opened.

mode : This is the C string containing a file access mode.
(next slide)

Return a FILE pointer on success, NULL on error.

The C library I/O APIs: fopen() (Cont.)

- **mode argument**
  - C string containing a file access mode.

Mode	Description
"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.
"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output).
"a+"	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file.
"wt+"	**write/update:** Create an **text** file and open it for update. (“w” for write , “t” for text)

The C library I/O APIs: fwrite()

- **fwrite**
  - Writes an array of count elements from the block of memory pointed by pointer to the current position in the stream.

```
#include <stdio.h>
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
```

```
EX: FILE *fp;
      char ch = 'a';
      int sByte = fwrite( &ch , sizeof(char), 4096, fp);
```

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.

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

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

Return number of bytes written on success , -1 on error.

The C library I/O APIs: fclose()

- **fclose**
  - Closes the file associated with the stream and disassociates it.

```
#include <stdio.h>
int fclose ( FILE * stream );
```

```
EX: FILE *fp;
      fclose(fp);
```

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

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

Example

```
#include<stdio.h>  #include<sys/types.h>  #include<sys/stat.h>
#include<fcntl.h>   #include<unistd.h>
#include<stdlib.h>    //for exit()
#include <string.h>  //for strlen()

int main(){
		FILE *fp;
		int size;
		char ch = 'a';  //the write data
		int element = 1;   //the byte writes

		fp = fopen("myfile.txt", "wt+");
		if (fp < 0){
				printf("Failed to open the file.\n");exit(1);
		}

		size = fwrite( &ch , sizeof(char), element, fp);
		//printf("Number of bytes written on success=%d \n\n", size);
		fclose(fp);
		return 0;
}
```

Outline

- **Regular file I/O by the system calls**
  - Kernel buffer (page cache)
  - I/O system calls
- **Regular file I/O by the C library APIs**
  - Application Buffer, C Library Buffer and Kernel Buffer
  - C library I/O APIs
- **Time function**
- **Homework Assignment #2**
- **Reference**

Time function

- **gettimeofday()**
  - Obtain the current time, expressed as seconds and microseconds.

```
#include<sys/time.h>
int gettimeofday(struct timeval *tp, struct timezone *tzp);
```

```
EX: struct timeval t_start,t_end;
       gettimeofday(&t_start, NULL);         //get start time
       gettimeofday(&t_end, NULL);           //get end time
       cost_time = t_end.tv_usec - t_start.tv_usec;
```

tp : Store the current time in the timeval structure pointed to by *tp*.

tzp : If *tzp* is not a null pointer, the behavior is unspecified.

Return 0 on success, -1 on error.

Outline

- **Regular file I/O by the system calls**
  - Kernel buffer (page cache)
  - I/O system calls
- **Regular file I/O by the C library APIs**
  - Application Buffer, C Library Buffer and Kernel Buffer
  - C library I/O APIs
- **Time function**
- **Homework Assignment #2**
- **Reference**

Homework Assignments #2

- Exercise 1 :

  Write a program that measure the times it takes to issue 4096,000 one-byte writes in each of two ways.

  - a. First, time how long it takes to use the POSIX system calls open(), write(), and close() directly.
  - b. Finally how long these writes take if the program uses the C stdio library calls (e.g., fopen(), fwrite(), and fclose()) instead.
  - Explain your results.

Homework Assignments #2

- Exercise 2 :

  Write a program that measure the times it takes to issue 1,000 4K-byte writes in each of two ways.

  4096

  - a. First, time how long it takes to use the POSIX system calls open(), write(), and close() directly.
  - b. Then see how long these writes take if the program uses the C stdio library calls (e.g., fopen(), fwrite(), and fclose()) instead.
  - Explain your results.

Homework Assignments #2

- Exercise 3 :

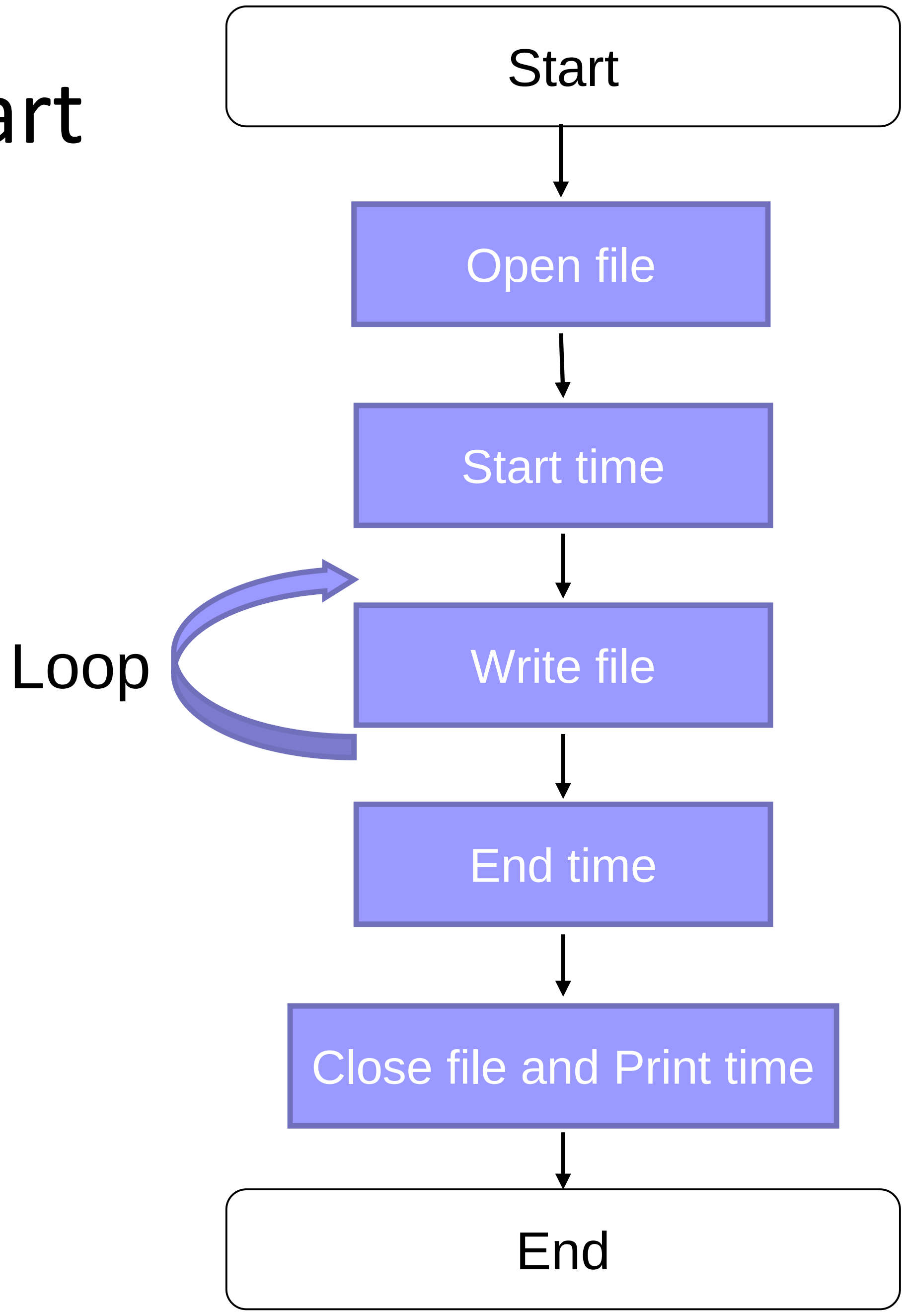
  Write a program that measure the times it takes to issue 500 8K-byte writes in each of two ways.

  - a. First, time how long it takes to use the POSIX system calls open(), write(), and close() directly.
  - b. Then see how long these writes take if the program uses the C stdio library calls (e.g., fopen(), fwrite(), and fclose()) instead.
  - Explain your results.

- Exercise 4 :

  Compare and explain the results from (1), (2), and (3).

Flowchart

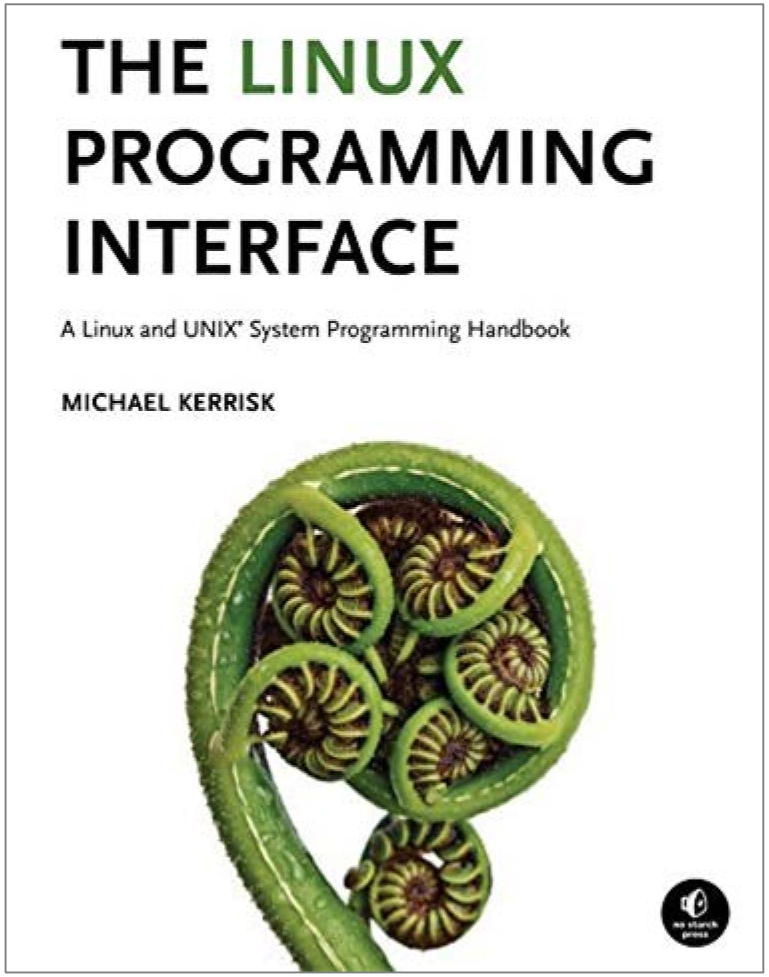

Start
Open file
Start time
Loop
Write file
End time
Close file and Print time
End

Outline

- **Regular file I/O by the system calls**
  - Kernel buffer (page cache)
  - I/O system calls
- **Regular file I/O by the C library APIs**
  - Application Buffer, C Library Buffer and Kernel Buffer
  - C library I/O APIs
- **Time function**
- **Homework Assignment #2**
- **Reference**

Reference

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



THE LINUX PROGRAMMING INTERFACE

A Linux and UNIX® System Programming Handbook

MICHAEL KERRISK

Turn in

- Deadline

  2020/04/28 PM.11:59:59
- Upload to iLearning
- File name
  - HW2_ID (e.g. HW2_4106056000)
    - Source code
      - .c file
    - Word

- If you don’t hand in your homework on time, your score will be deducted 10 points every day.

TA

- Name：Yun-Jen, Lee
- Email：yunjen.lee@gmail.com
  - Title format : OS HW#2 - [your name]
- Lab: OSNET(1001)