Operating System Labs

Yuanbin Wu
cs@ecnu

Annoucement

- Next Monday (28 Sept):
  - We will have a lecture @ 4-302, 15:00-16:30
  - DON'T GO TO THE LABORATORY BUILDING!

- TA email update:
  - ecnucchuang@163.com → ecnucchuang@126.com

Operating System Labs

- Introduction of I/O operations
- Project 1
  - Sorting

Operating System Labs

- Manipulate I/O
  - System call
    - File descriptor
    - No buffering
  - Standard library
    - FILE object
    - Buffering

Operating System Labs

- Manipulate I/O
  - System call
    - File descriptor
  - Standard library
    - FILE object
    - Buffer/non-buffer

I/O System Calls

- 5 basic system calls
  - open(), read(), write(), lseek(), close()
- I/O without buffering
- File sharing
  - understand descriptor
- Other
  - dup(), fcntl(), sync(), fsync(), ioctl()

I/O System Calls

- File descriptor
  - Allocated when open a file
  - “ID” of the file in the process
- Default
  - 0 (STDIN_FILENO): standard input
  - 1 (STDOUT_FILENO): standard output
  - 3 (STDERR_FILENO): standard error

I/O System Calls

- Open files:

```
# include <fcntl.h>

int open(const char *pathname, int o_flag, …  );
```

- Return value
  - Success: file descriptor
  - Failed: -1
- o_flag:
  - O_RDONLY, O_WRONLY, O_RWWR
  - Options:
    - O_APPEND, O_CREAT, O_TRUNC, ...

I/O System Calls

- Open files
  - File descriptors: the smallest one available
  - Examples

```
int main (int argc, char **argv)
{
    int fd = open(“foo”, O_RDONLY);
    printf(“%d”, fd);
}
```

```
int main (int argc, char **argv)
{
    close(0);
    int fd = open(“foo”, O_RDONLY);
    printf(“%d”, fd);
}
```

I/O System Calls

- Open files
  - STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO
  - opened by the OS when creating a process

I/O System Calls

- Close files

```
# include <unistd.h>

int close(int filedes);
```

- Return
  - Success: 0
  - Failed: -1

I/O System Calls

- File Position

```
# include <unistd.h>

off_t lseek(int filedes, off_t offset, int whence);
```

- “Current file offset”:
  - An offset (in byte) to the beginning of the file
- whence:
  - SEEK_SET, SEEK_CUR, SEEK_END

I/O System Calls

- Read files

```
# include <unistd.h>

int read(int filedes, void *buf, size_t nbytes);
```

- Return:
  - Success: number of bytes read (0, if EOF)
  - Failed: -1
- Return < size
  - EOF
  - Read from terminal (stdin), one line
  - ...

I/O System Calls

- Write files

```
# include <unistd.h>

int write(int filedes, const void *buf, size_t nbytes);
```

Return:

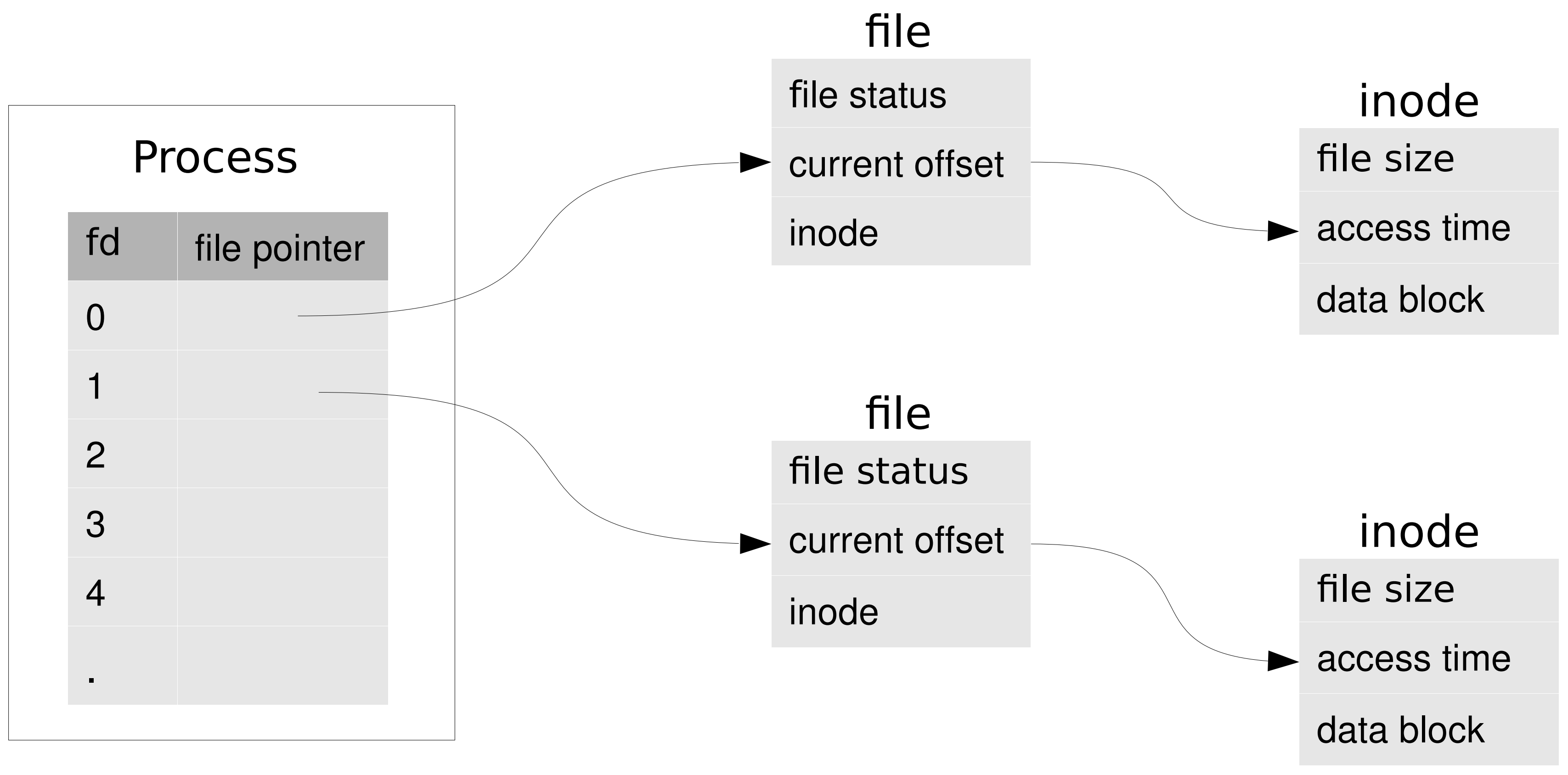
- Success: number of bytes write
- Failed: -1

I/O System Calls

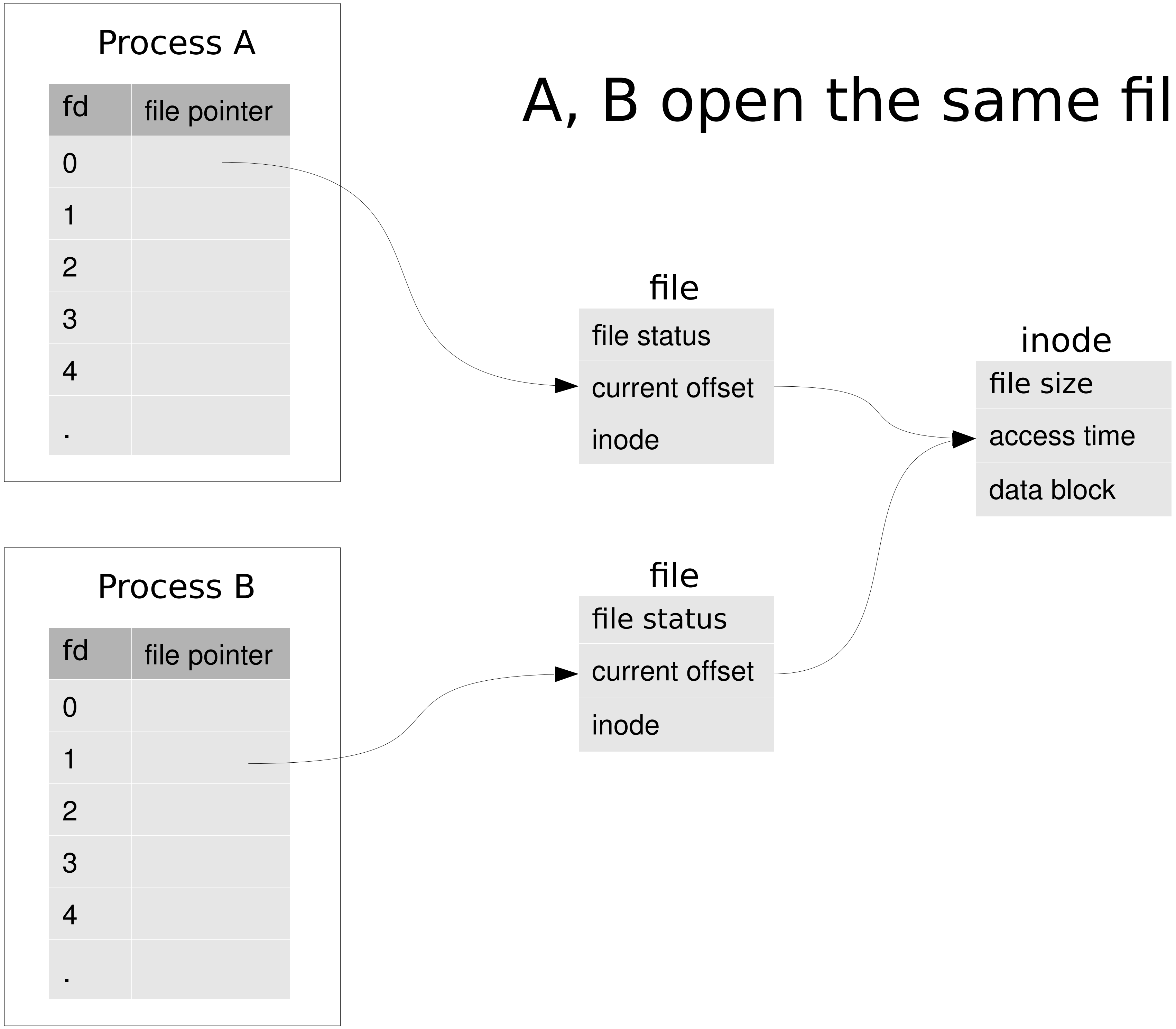
- I/O without buffer
  - No (user space) buffer
    - read(), write(): no buffer in user space
    - Do have buffer in kernel space
  - Let's do some coding
  - Buffering do matter!
    - printf, scanf in standard I/O library are buffered

I/O System Calls

- File descripter revisit



A, B open the same file



I/O System Calls

- File sharing
  - Automic operations
  - Example: open(“file”, O_WRONLY | O_APPEND)
  - Two process A, B run the same code, what will happen?

```
if (lseek(fd, O, SEEK_END) < 0)
     perror(“lseek”);

if (write(fd, buf, 100) < 100)
     perro(“write”);
```

I/O System Calls

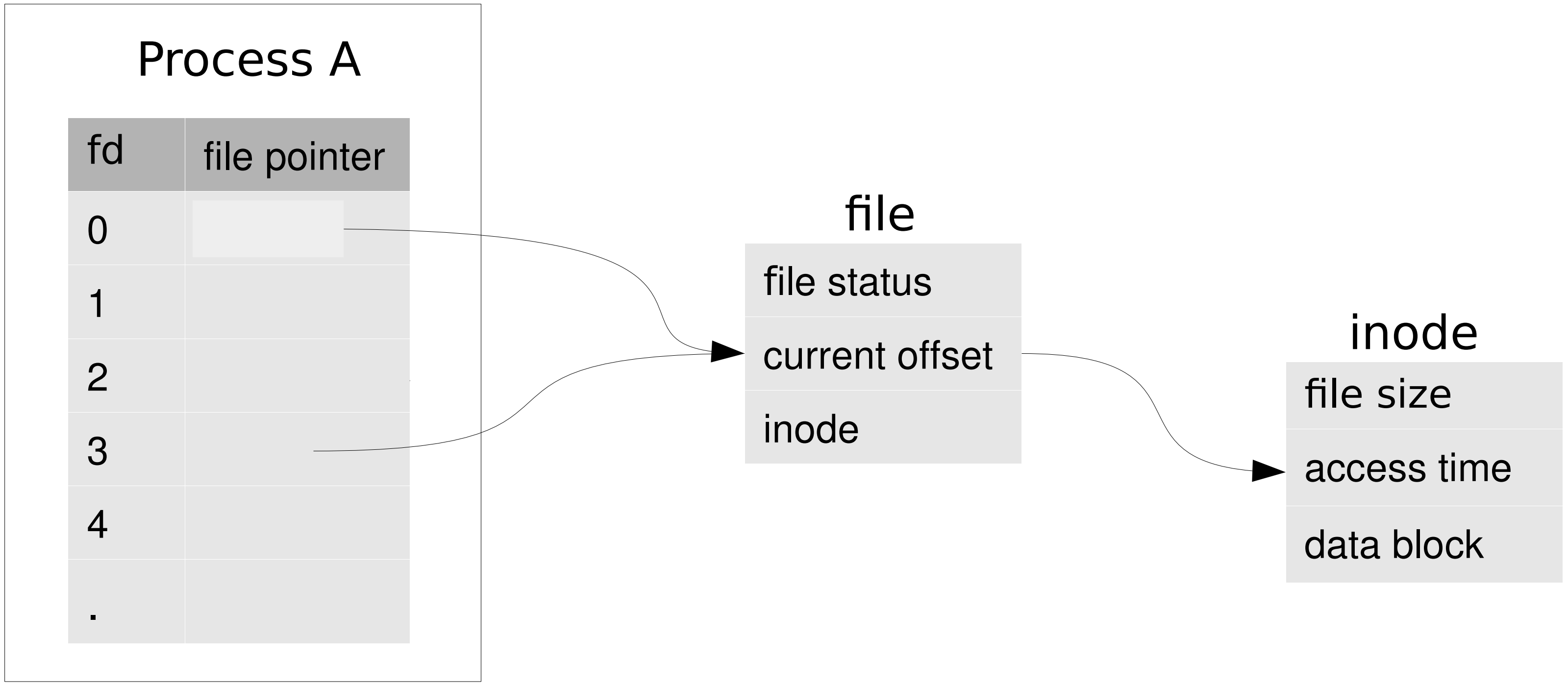
- Duplicate file descriptor

```
# include <unistd.h>

int dup2(int fd, int fd2);
```

- set “fd2” point to the same file of “fd”
- Return
  - Success: fd
  - Failed: -1

```
// fd 0 has been closed
dup2(3, 0);
```



a file with multiple file descriptors

I/O System Calls

- Other system calls
  - sync() / fsync(): enque the kernel buffer
  - fcntl(): change file (opened) attributes
  - ioctl(): other methods

I/O System Calls

- Summary
  - File descriptor
  - open, close, read, write, lseek, dup
  - File sharing

Operating System Labs

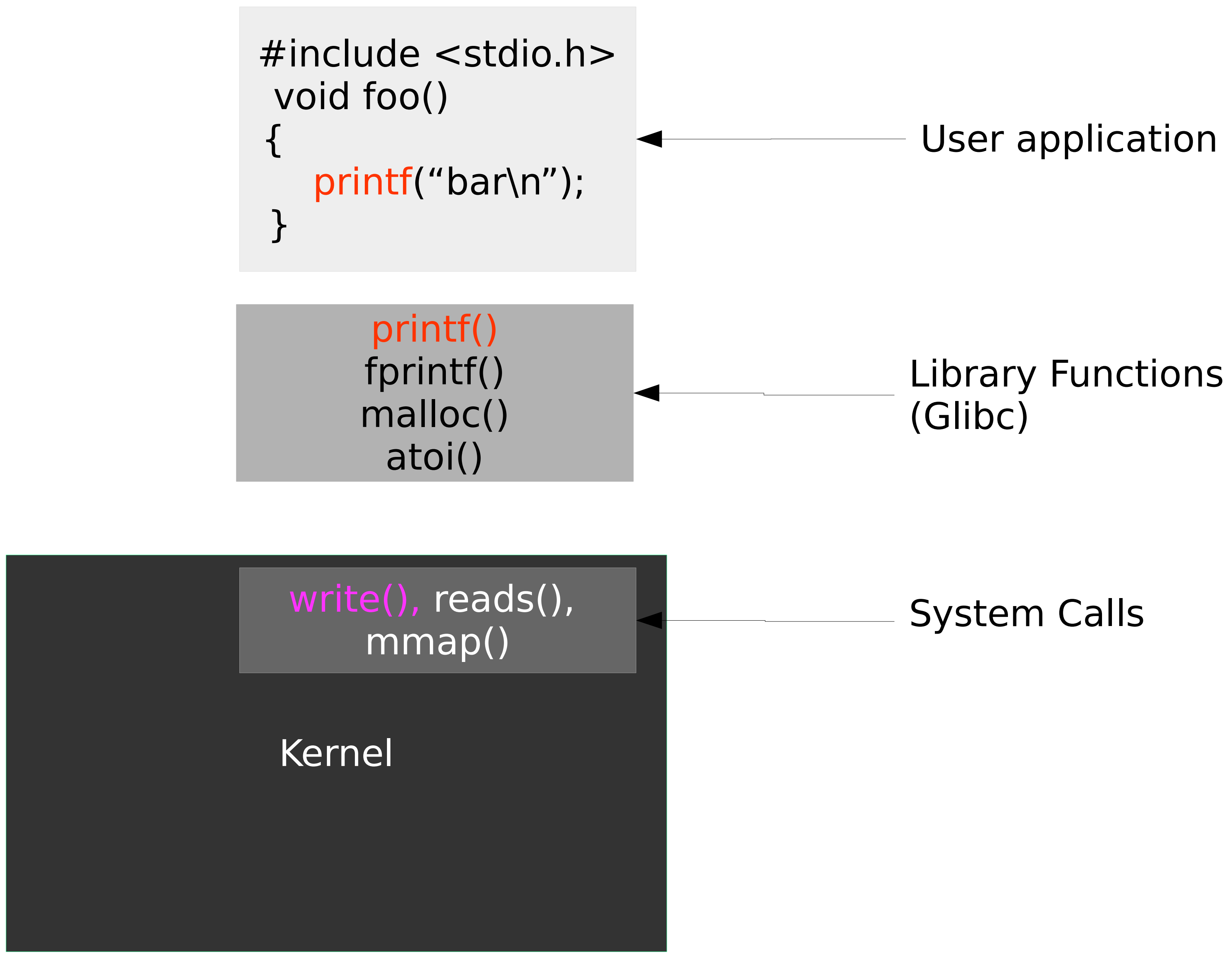
- Manipulate I/O
  - System call
    - File descriptor
    - No buffering
  - Standard library
    - FILE object
    - Buffering

Standard I/O Library

- #include <stdio.h>
  - FILE object (structure)
  - Buffering
  - Formatted I/O

Standard I/O Library

- Recall:


```
#include <stdio.h>
 void foo()
{
    printf("bar\n");
}
```

Standard I/O Library

```
# include <fcntl.h>

int main (int argc, char **argv)
{
	int fd = open(“foo”, O_RDONLY);
}
```

```
# include <stdio.h>

int main (int argc, char **argv)
{
	FILE* fp = fopen(“foo”, “r”);
}
```

- Stream and FILE object
  - A wrapper of file descriptor
  - More information: buffer, error info.

Standard I/O Library

- Buffering
  - stdio provide a “standard I/O buffer” (user space)
- Three types of buffering
  - Full buffered
    - Performs I/O when the buffer is full
  - Line buffered
    - Performs I/O when encounter a newline
  - Unbuffered
    - Performs I/O immediately, no buffer

Standard I/O Library

- Three types of buffering: cases
  - Standard error is unbuffered
  - A stream is line buffered if it refers to terminal device, otherwise full buffered

- Write “standard I/O buffer” to disc:

```
# include <stdio.h>

int fflush(FILE *fp);
```

Standard I/O Library

- Open streams

```
# include <stdio.h>

FILE *fopen(const char* path, const char * type);
```

- Type: “r”, “w”, “a”.. .
- Return
  - Failed: NULL

Standard I/O Library

- Character-at-a-time I/O

```
# include <stdio.h>

int getc(FILE *fp);
int fgetc(FILE *fp);

int putc(FILE *fp);
int fputc(FILE *fp);
```

Standard I/O Library

- Line-at-a-time I/O

```
# include <stdio.h>

char* fgets(char *buf, int n, FILE *fp);
char* gets(char *buf);

int fputs(char *str, FILE *fp);
int puts(char *str);
```

Standard I/O Library

- Direct I/O

```
# include <stdio.h>

size_t fread(void *ptr, size_t size, size_t, nobj, FILE *fp);
size_t fwrite(void *ptr, size_t size, size_t, nobj, FILE *fp);
```

Standard I/O Library

- Standard I/O efficiency
  - Recall: buffering in system calls
  - Let's do some coding again

Standard I/O Library

- Formatted I/O
  - printf, fprintf, scanf

Standard I/O Library

- Summary
- #include <stdio.h>
  - FILE object (structure)
  - Buffering
  - Formatted I/O

Introduction of I/O Operations

- Summary
  - System call
    - File descriptor
    - No buffering
  - Standard library
    - FILE object
    - Buffering

Project 1

- Sorting

Annoucement

- Next Monday:
  - We will have a lecture @ 4-302, 15:00-16:30
  - DON'T GO TO THE LABORATORY BUILDING!

- TA email update:
  - ecnucchuang@163.com → ecnucchuang@126.com