Project #2: User Program 2

[CSE4070]

Fall 2019

MinGyo Jung

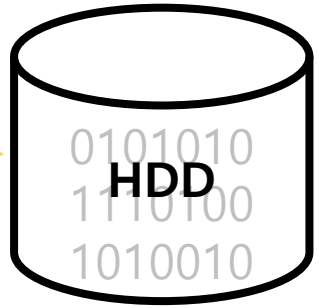
Contents

What is File system?

```
                :~$ ls -l
total 28
-rw-r--r-- 1                186 Nov  1 00:31 a.c
-rw-r--r-- 1               3319 Nov  1 00:32 a.txt
drwxr-xr-x 2               4096 Oct 31 02:54 Desktop
drwxr-xr-x 2               4096 Oct 31 02:54 Downloads
drwxr-xr-x 2               4096 Oct 31 02:58 example
drwxr-xr-x 2               4096 Oct 31 19:50 test
drwxr-xr-x 7               4096 Nov  1 00:30 tmp
```

ext4

User




```c
#include <stdio.h>

int main(){
    ......

    FILE* fp = fopen("test/input.txt", "r");

    ......
}
```

Access

What to do?

- **Implement system calls about file system**
  - System calls
    - create, remove, open, close, filesize, read, write, seek, tell
  - See Pintos manual p.29~32 for detail system call implementation

Base File System

Base File System

- **To implement system call about file system,**
  we need to understand base file system of Pintos
  - We need to interface to the file system code. So, we need to know API usage.
  - Pintos provides simple but complete file system.
    (Look `filesys.h` and `file.h` at `filesys/`)
  - Don't need to understand whole structure of file system.
  - No need to modify the file system code for this project.

Base File System

- **Limitation of Base File System (See also p.23-24):**
  - No internal synchronization
  - File size is fixed at creation time
  - File data is allocated as a contiguous range of sectors on disks
  - No subdirectories
  - File name length limitation (up to 14 characters)

Base File System

- **Usage about file system on project 2**
  - There is no sub-directories.
  - We assume that we work on root directory (/) of file system.

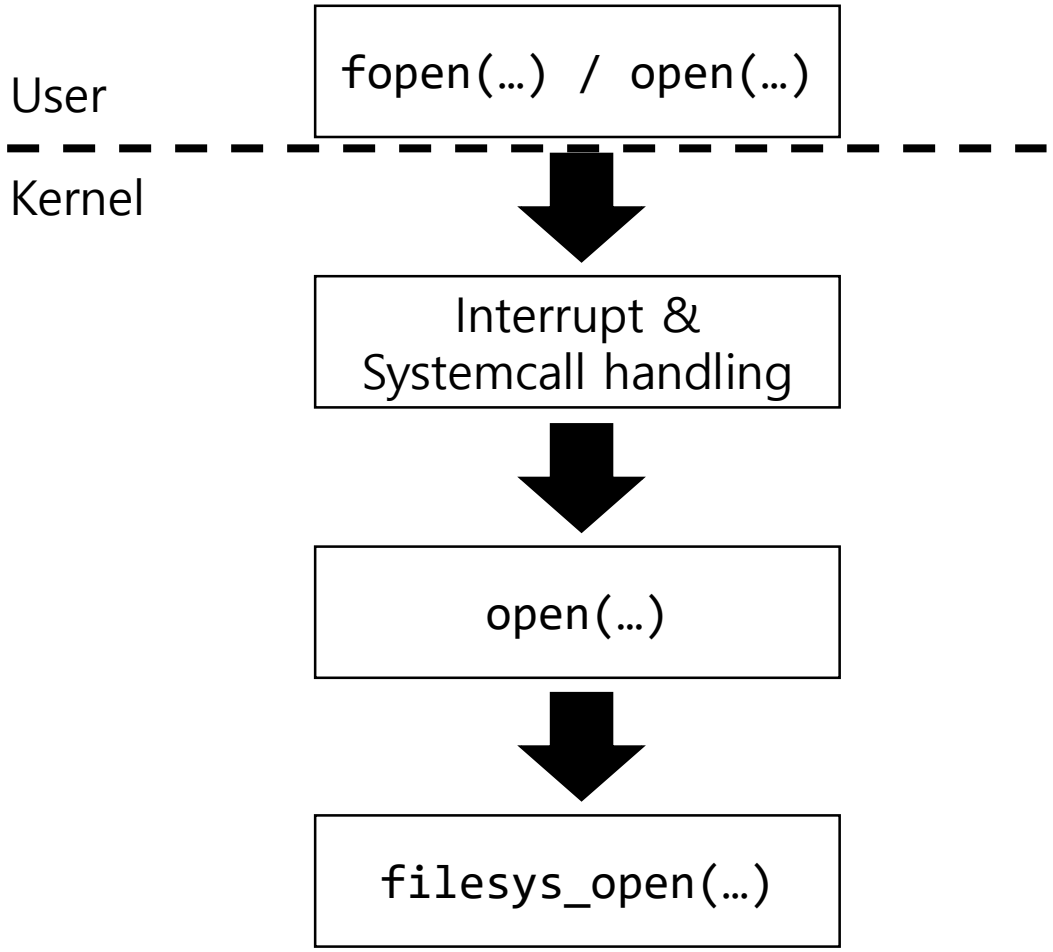
- **File access at Kernel**
  - Can access through `struct file` (See `filesys/file.c`)
  - (At this project, it doesn't need to access on inode level)

```
/* An open file. */
struct file
  {
    struct inode *inode;        /* File's inode. */
    off_t pos;                  /* Current position. */
    bool deny_write;            /* Has file_deny_write() been called? */
  };
```

File System Call Implementation

File System Call Implementation

- **Basic file system usage interfaces:**



- Call `fopen(or open)` at user program
- Interrupt 0x30 and call systemcall handler
- Call open systemcall

We need to implement

- At internal implementation of open, it calls `filesys_open` function and open that file

File System Call Implementation

- **Example in Pintos**

```c
/* Open a file. */

#include <syscall.h>
#include "tests/lib.h"
#include "tests/main.h"

void
test_main (void)
{
  int handle = open ("sample.txt");
  if (handle < 2)
    fail ("open() returned %d", handle);
}
```

```c
int
open(const char *file)
{
  ......

  struct file * f = filesys_open(...);

  ......
}
```

```c
struct file *
filesys_open (const char *name)
{
  struct dir *dir = dir_open_root ();
  struct inode *inode = NULL;
  if (dir != NULL)
    dir_lookup (dir, name, &inode);
  dir_close (dir);

  return file_open (inode);
}
```

System call acts like interfaces for calling `file_xxx`

File System Call Implementation

- **File Descriptor (Similar to `FILE *` of standard C)**
  - Return value of open, `create`
  - At Pintos, each thread get and manage independent file descriptor.
  - File descriptor of STDIN, STDOUT, STDERR
    - STDIN = 0
    - STDOUT = 1
    - (At Linux or other OSes, STDERR = 2)

File System Call Implementation

- **`read` and `write` with `stdin`/`stdout`**
  - Read from stdin – read(0)
    - Use `uint8_t input_getc(void)` (`devices/input.c`)
  - Write to stdout – write(1)
    - Use `void putbuf(…)` (`lib/kernel/console.c`)

Useful APIs

- **filesys/filesys.h**

```
bool filesys_create (const char *name, off_t initial_size);
struct file *filesys_open (const char *name);
bool filesys_remove (const char *name);
```

- **filesys/file.h**

```
/* Opening and closing files. */
struct file *file_open (struct inode *);
void file_close (struct file *);

/* Reading and writing. */
off_t file_read (struct file *, void *, off_t);
off_t file_write (struct file *, const void *, off_t);

/* Preventing writes. */
void file_deny_write (struct file *);
void file_allow_write (struct file *);

/* File position. */
void file_seek (struct file *, off_t);
off_t file_tell (struct file *);
off_t file_length (struct file *);
```

Kernel Functions

Denying Writes to Executable files

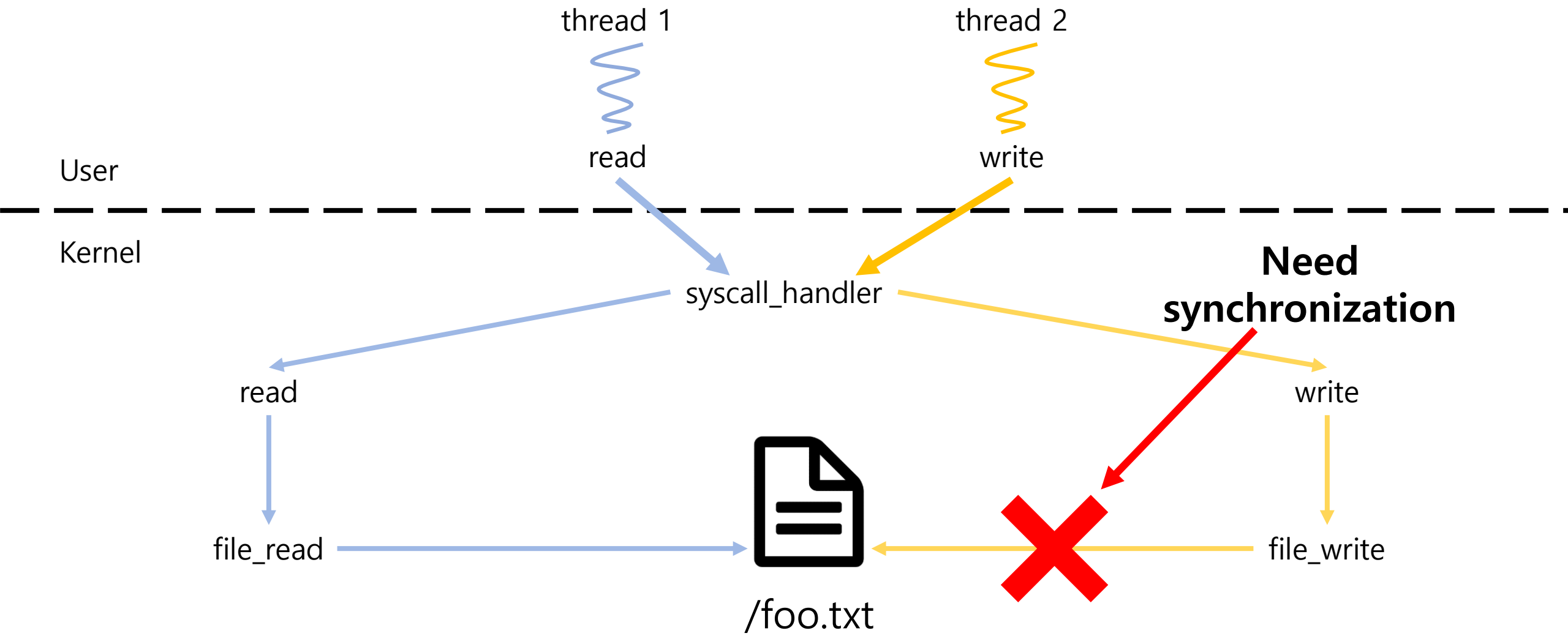
- **Process is excuted after load to memory.**
  - So, removing executable file may not be a problem after executed.
- **But Pintos doesn't want to delete executable file of running program.**
- **These file system function may be useful for this problem:**
  - `void file_deny_write(struct file *)` (filesys/file.c)
  - `void file_allow_write(struct file *)` (filesys/file.c)

Protect Critical Section

- **Critical Section**
  - N Processes all competing to use some shared data
  - Each Process has a code segment, called critical section, in which the shared data is accessed
  - Problem: Ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section

- **Using test case of Pintos, check code segment that should be considered as critical section**
  - Protect this code segment using synchronization APIs.
  - ex) syn-read, syn-write

Critical Section – Example in Pintos



Synchronization APIs (See also p.67-68)

- **Lock**

```
/* Lock. */
struct lock
  {
    struct thread *holder;      /* Thread holding lock (for debugging). */
    struct semaphore semaphore; /* Binary semaphore controlling access. */
  };

void lock_init (struct lock *);
void lock_acquire (struct lock *);
bool lock_try_acquire (struct lock *);
void lock_release (struct lock *);
bool lock_held_by_current_thread (const struct lock *);
```

Synchronization APIs (See also p.67-68)

- **Semaphore**

```
/* A counting semaphore. */
struct semaphore
  {
    unsigned value;             /* Current value. */
    struct list waiters;        /* List of waiting threads. */
  };

void sema_init (struct semaphore *, unsigned value);
void sema_down (struct semaphore *);
bool sema_try_down (struct semaphore *);
void sema_up (struct semaphore *);
void sema_self_test (void);
```

Evaluation and Submission

Evaluation

1. Rest of 76 tests remained from project 1 will be graded. (Total 55 of 76 tests)
   - Refer to the test case list in the next slide
2. Total score is 100 which consists of 80 for test cases and 20 for documentation
3. Grading script provided by Pintos will be used.
   - `make grade` or `make check` in src/userprog
4. Refer to “grade” and “results” files in src/userprog/build after grading
   - “grade” file is only created when you use `make grade`
5. Test cases are classified in functionality / robustness / base file system (filesys base)
6. Refer to the followings for checking each test case’s point based on the test type
   - tests/userprog/Rubric.functionality
   - tests/userprog/Rubric.robustness
   - tests/filesys/base/Rubric

Evaluation: Test Cases (55 tests)

User Program

Functionality		
No.	Name	Point
1	create-empty	3
2	create-long	3
3	create-normal	3
4	create-exists	3
5	open-missing	3
6	open-normal	3
7	open-twice	3
8	read-normal	3
9	read-zero	3
10	write-normal	3
11	write-zero	3
12	close-normal	3

Functionality		
No.	Name	Point
13	rox-simple	3
14	rox-child	3
15	rox-multichild	3
Total Score		**45**
Robustness		
1	close-stdin	2
2	close-stdout	2
3	close-bad-fd	2
4	close-twice	2
5	read-bad-fd	2
6	read-stdout	2
7	write-bad-fd	2
8	write-stdin	2

Robustness		
No.	Name	Point
9	multi-child-fd	2
10	create-bad-ptr	3
11	exec-bad-ptr	3
12	open-bad-ptr	3
13	read-bad-ptr	3
14	write-bad-ptr	3
15	create-bound	3
16	open-boundary	3
17	read-boundary	3
18	write-boundary	3
19	create-null	2
20	open-null	2
21	open-empty	2

Evaluation

User Program

Robustness		
No.	Name	Point
22	bad-read	1
23	bad-read2	1
24	bad-write	1
25	bad-write2	1
26	bad-jump	1
Total Score		56
No-vm		
1	multi-oom	1
Total Score		1

Filesys

Base		
No.	Name	Point
1	sm-create	1
2	sm-full	2
3	sm-random	2
4	sm-seq-block	2
5	sm-seq-random	3
6	lg-create	1
7	lg-full	2
8	lg-random	2
9	lg-seq-block	2
10	lg-seq-random	3

Base		
No.	Name	Point
11	syn-read	4
12	syn-write	4
13	syn-remove	2
Total Score		30

# of Total Tests	55

Evaluation

- **Total score:**

$$\left(\frac{\text{Functionality}}{45} \times 35 + \frac{\text{Robustness}}{56} \times 25 + \frac{\text{no-vm}}{1} \times 10 + \frac{\text{filesys(base)}}{30} \times 30\right) / 100 \times 80$$

- **Documentation**
  - Use the document file uploaded on e-class.
  - Documentation accounts for 20% of total score.
    (Development 80%, Documentation 20%)

Submission

- **Before submission, check that you performed the following:**

1. **Make Clean** before compressing
2. Submission form (Refer to the next slide)
3. Once you copy other's codes, you will get **F grade**

Submission

1. **Team project**
2. **Due date: 2019. 11. 02 (Sat) 23:59**
3. **Submission**
   - The form of submission file is as follows:

Name of compressed file	Example (project 2, Group #7)
os_prj2_##.tar.gz	os_prj2_07.tar.gz

   - Only one person of group should submit the file.
   - **You should also submit the design document in hardcopy (AS712A).**
     (Due date of hardcopy is same as the compressed file)

Submission

- **Contents**
  - ① Pintos source codes
  - ② document: **document.doc** or **document.docx** (Other format is not allowed such as .hwp)
- **Form and way to submit**
  1) How to make tar.gz file
     - Run 'make clean' in pintos/src
     - Make new directory with your group number
     - Copy 'pintos' directory (not pintos/src) into the new directory
     - Copy the document file into the new directory
     - Compress the new directory into os_prj2_##.tar.gz **(tar -zcf os_prj2_[Group#].tar.gz ./[Group#])**
       - ✓ **For example, if your group number is 7, tar -zcf os_prj2_07.tar.gz ./07**
       - ✓ You should use -zcf options for using tar
  2) Way to submit: Upload the tar.gz file to e-class
     - **Submit Hardcopy to AS712A (You should submit softcopy and hardcopy both, if not 3% of point will be deducted)**
  3) Due date: 2019. 11. 02. 23:59

❖ **5% of point will be deducted for a wrong form and way to submit**

❖ **Late submission is allowed up to 3 days (~11/6) and 10% of point will be deducted per day**

Project Schedule

Projects	Points	Contents	Periods	Lectures
Project 0-1	1	Installing Pintos	9/16 – 9/22	Manual will be provided
Project 0-2	3	Pintos Data Structures	9/21 – 10/6	9/21 (Sat.)
Project 1	6	User Programs (1)	10/5 – 11/3	10/5 (Sat.)
Project 2	**4**	**User Programs (2)**	**11/2 – 11/17**	**11/2 (Sat.)**
Project 3	6	Threads	11/16 – 12/8	11/16 (Sat.)

※ Once you copy other's codes, you will get F grade