Project #5: Pintos Filesystem

[CSE4070]

Instructor
Prof. Youngjae Kim

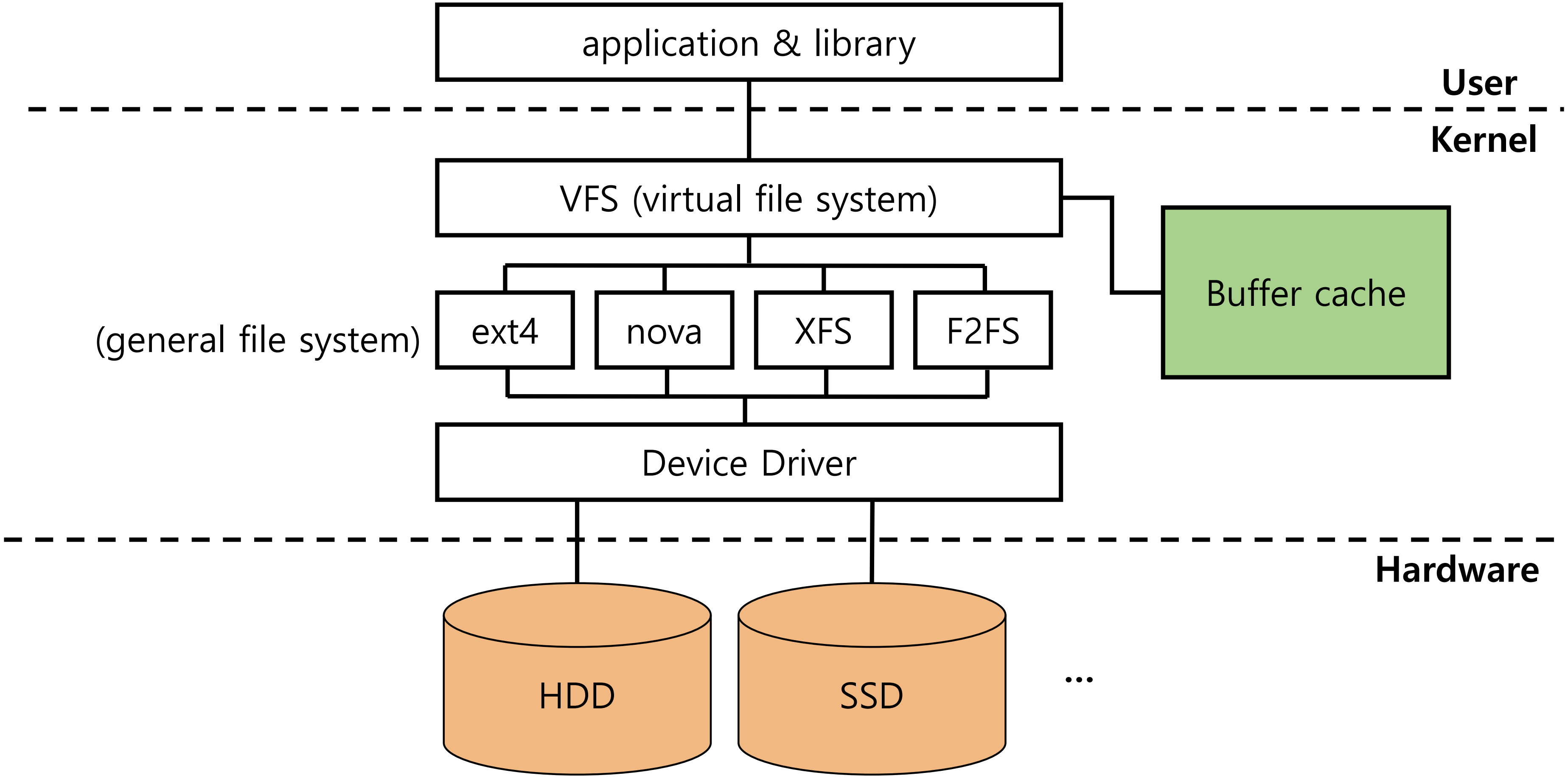
Teaching Assistants
Yonghyeon Cho (01)
Jungwook Han (02)

Fall 2021

서강대학교
SOGANG UNIVERSITY

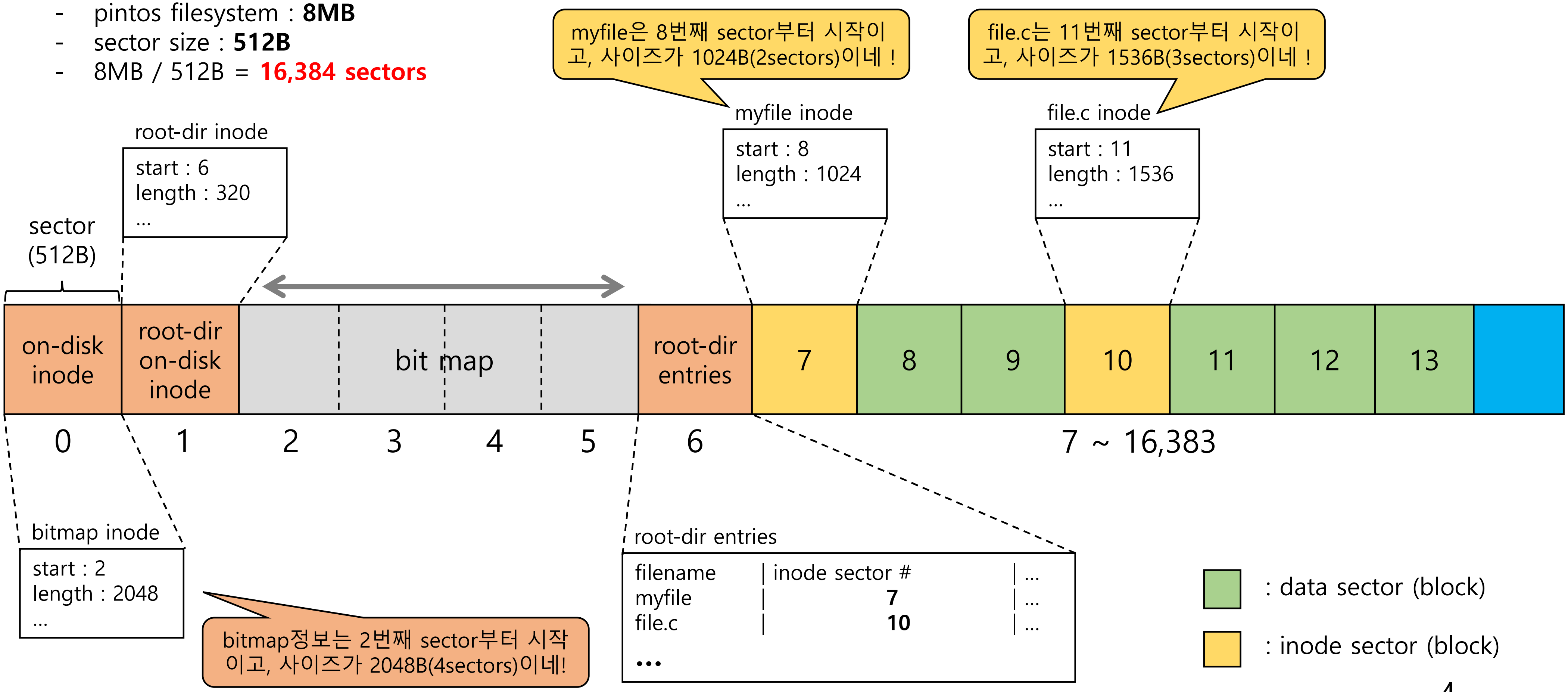
Filesystem

Pintos Filesystem (overview)



Pintos Filesystem (on-disk layout)

- pintos filesystem : **8MB**
- sector size : **512B**
- 8MB / 512B = **16,384 sectors**

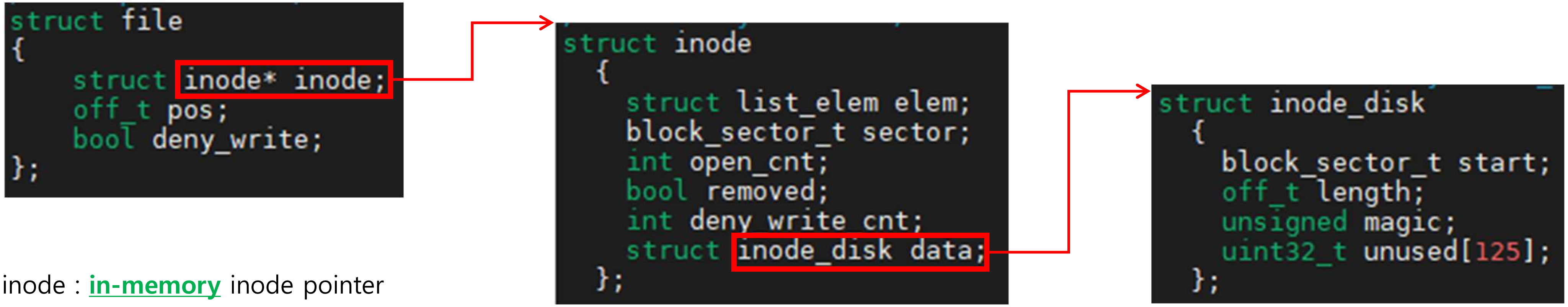


filename	inode sector #	
myfile	**7**	…
file.c	**10**	…
…		

🟩 : data sector (block)

🟨 : inode sector (block)

Pintos Filesystem (file structure)



```
struct file
{
    struct inode* inode;
    off_t pos;
    bool deny_write;
};
```

```
struct inode
  {
    struct list_elem elem;
    block_sector_t sector;
    int open_cnt;
    bool removed;
    int deny_write_cnt;
    struct inode_disk data;
  };
```

```
struct inode_disk
  {
    block_sector_t start;
    off_t length;
    unsigned magic;
    uint32_t unused[125];
  };
```

file

- inode : **in-memory** inode pointer
- pos : file offset (position)
- deny_write : write accessible or not

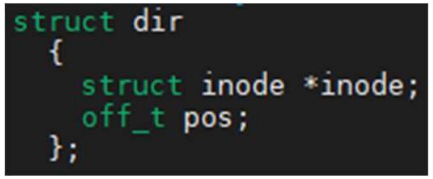
In-memory inode

- sector : block number with inode
- data : disk_inode data
- remove : file removed or not

On-disk inode

- start : start block number of file data
- length : size of file (Byte)

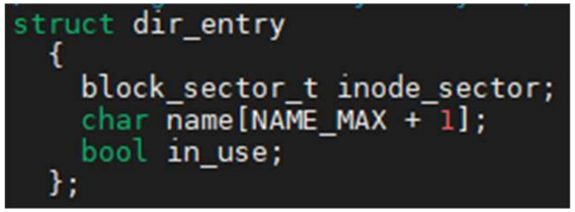
Pintos Filesystem (directory & bitmap)



```
struct dir
  {
    struct inode *inode;
    off_t pos;
  };
```

dir structure (**in-memory**)

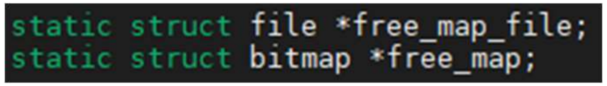
- inode : directory inode pointer
- pos : directory entry offset



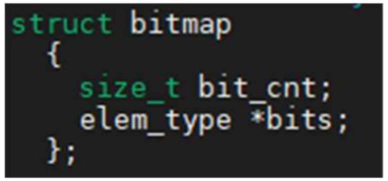
```
struct dir_entry
  {
    block_sector_t inode_sector;
    char name[NAME_MAX + 1];
    bool in_use;
  };
```

dir entry structure (**on-disk**)

- inode_sector : inode's disk block number
- name : MAX length : 14
- in_use : dir_entry in use or not



```
static struct file *free_map_file;
static struct bitmap *free_map;
```

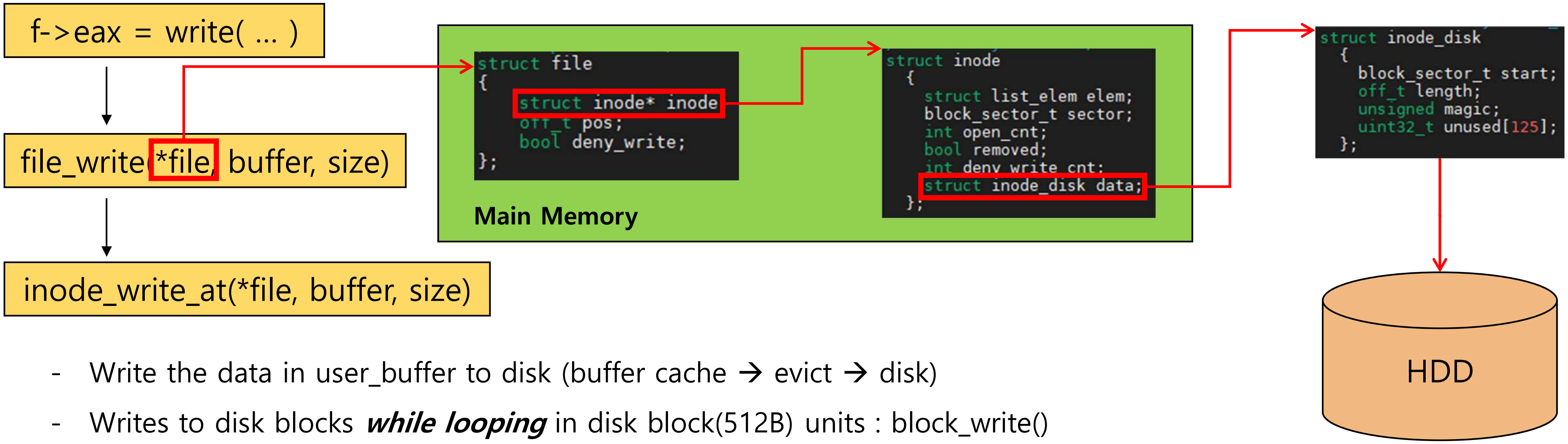


```
struct bitmap
  {
    size_t bit_cnt;
    elem_type *bits;
  };
```

bitmap structure

- free_map_file : store bitmap at **disk** in file format
- bit_cnt : disk block number of filesystem

Pintos Filesystem (write code path)



f->eax = write(…)

file_write(*file, buffer, size)

inode_write_at(*file, buffer, size)

```
struct file
{
    struct inode* inode
    off_t pos;
    bool deny_write;
};
```

```
struct inode
  {
    struct list_elem elem;
    block_sector_t sector;
    int open_cnt;
    bool removed;
    int deny_write_cnt;
    struct inode_disk data;
  };
```

```
struct inode_disk
  {
    block_sector_t start;
    off_t length;
    unsigned magic;
    uint32_t unused[125];
  };
```

- Write the data in user_buffer to disk (buffer cache → evict → disk)
- Writes to disk blocks ***while looping*** in disk block(512B) units : block_write()
    - byte_to_sector() : Disk block number to write data to
    - sector_ofs : Disk block offset to write data to
- In the case of partial write, read the target block and store in the bounce buffer

Pintos Filesystem (baseline) : 현재 pintos의 한계점

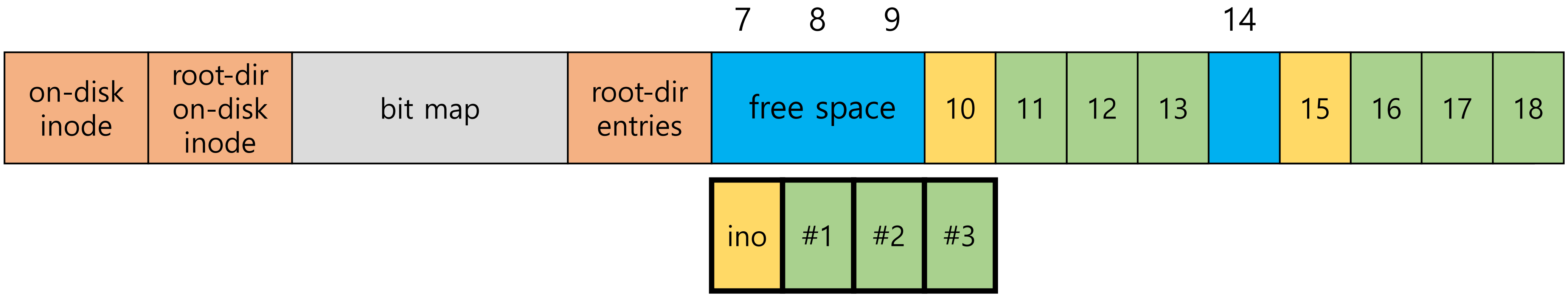
- external fragmentation : N-block file cannot be allocated even though N-blocks are free.
  - Each inode is stored in 1-disk sector, limiting the number of block pointers that is can contain
  - eleminate this problem by modifying on-disk inode structure
    (using direct, indirect, doubly indirect blocks and bitmap)
  - Use **doubly-indirect block**
- The filesize is specified when the file is created
  - Implement **file growth**
  - **Indexed inode** make file growth possible whenever free space is available
- No subdirectories
  - All files live in **a single directory (root dir)**.
  - Make this to allow directory entries to point to files or to other directories.
  - **14 char name** : Directory, File의 이름은 14글자로 지정 (늘리는 것은 optional!!)
  - Dir, File자체의 name length는 14미만이지만, **full path에 대해서는 처리**를 해줘야 한다.
    Example : /home/myname/pintos/src/filesys/file.c (14보다 길이가 길다.)

Pintos Filesystem (working area)

- filesystem translates these calls(creat, open, close, read, write, …) into disk operations.
- 8MB filesystem
- Working in pintor/src/filesys directory
- **fsutil.c** : simple utilities for the file system that are accessible from the kernel command
- **filesys.c/h** : top level interface to the file system
- **directory.c/h** : translates file names to inodes. (directory structure is stored as a file)
- **inode.c/h** : manage the data structure representing the file's on-disk layout
- **free-map.c/h** : mapping disk sector with bitmap (use lib/kernel/bitmap.c/h functions)
- **cache.c/h** : we need to make source code file by our own and implement on it.

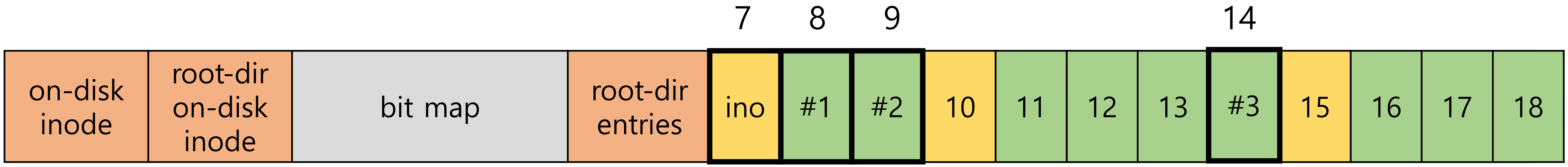
Extensible Files (External Fragmentation)

- When inode_create or write/read, follow the existing fragmentation
- To solve the problem, we need to make each data block accessible from the inode.
  - Modify the on-disk inode structure (direct, indirect, doubly indirect index structure)
  - Any design that does not suffer from external fragmentation is possible.
  - Indexed inodes allow files to grow whenever free space is available. (file growth)



Extensible Files (External Fragmentation)

- When inode_create or write/read, follow the existing fragmentation
- To solve the problem, we need to make each data block accessible from the inode.
  - Modify the on-disk inode structure (direct, indirect, doubly indirect index structure)
  - Any design that does not suffer from external fragmentation is possible.
  - Indexed inodes allow files to grow whenever free space is available. (file growth)



7 8 9 14

on-disk inode | root-dir on-disk inode | bit map | root-dir entries | ino | #1 | #2 | 10 | 11 | 12 | 13 | #3 | 15 | 16 | 17 | 18

Subdirectory

- Pintos filesystem baseline only have root directory and all files are under root directory
- **Modify the directory entry to point to a file or other directory.**
- The default file system has a 14-character limit on file names. (optional extend or not)
- The full path filename must be much longer than 14 characters.
  (ex. /home/cse20201234/pintos/src/…)
- Update existing system calls to use absolute or relative pathnames (consider /, ., ..)
- A directory can only be deleted if it contains no files or subdirectories (except '.' and '..').
- If you open a file, you must close it properly. (open_cnt = 0 → write back to disk)
  - check the function inode_open() or inode_reopen()
- A directory can also be opened. Open directories must not be removed.
- **persistency**
  - Even after booting again after termination, data should be well recorded.

Subdirectory (Syscall API)

(**src/filesys/directory.c, directory.h** 참고)

- open : update the open syscall to open directories in addition
- remove : update the remove syscall to delete empty directories in addition
- bool chdir (const char *dir)
  : change the current working directory of the process to dir(param).
  (relative or absolute path)
- bool mkdir (const char *dir)
  : create the directory name “dir” (relative or absolute path)
- bool readdir (int fd, char *name)
  : read a directory entry from file descriptor fd, which must represent a directory
- bool isdir (int fd)
  : return true if fd represent a directory, false if it represent an file.
- bool inumber (int fd)
  : return the inode number of the inode associated with fd.

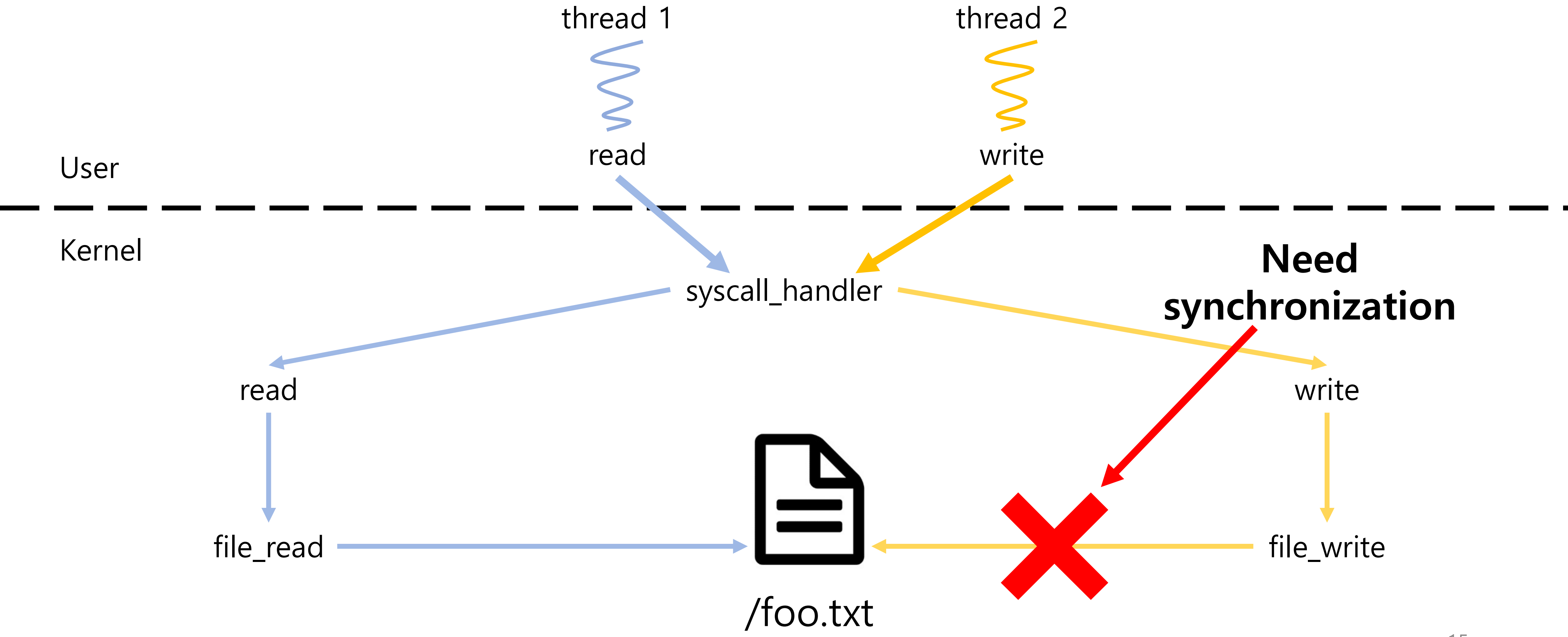
You can figure out more detail in Pintos Manual p53.

Synchronization (Protect Critical Section)

- Critical Section
  - N Processes all competing to access to same file.
  - Each Process has a code segment, called critical section, in which the shared data is accessed
  - Problem: Ensure that when one process is modifying the specific file,
    no other process can access this file (critical section)

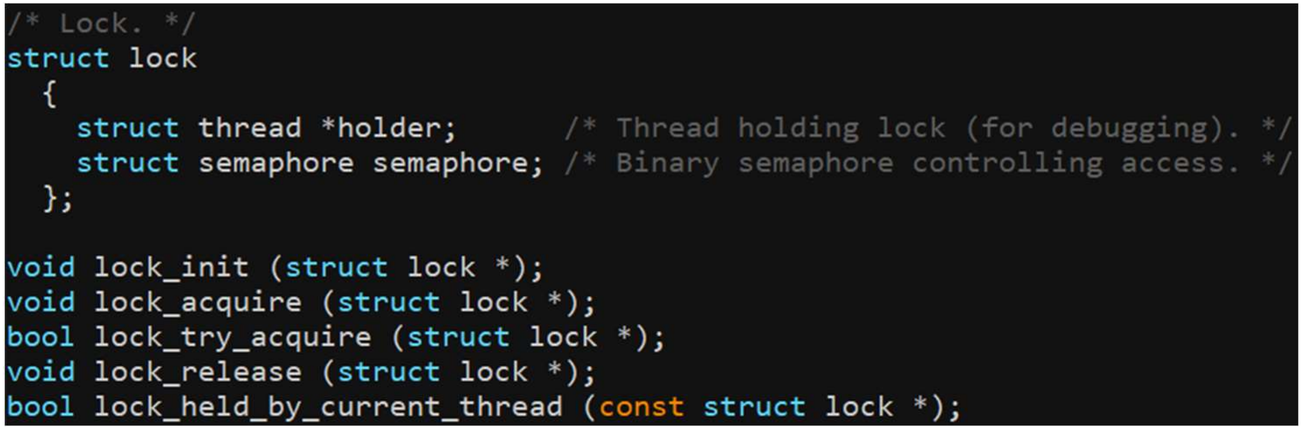
- Using test case of Pintos, check code segment that should be considered as critical section
  - Protect this code segment using synchronization APIs.
  - ex) syn-read, syn-write
  - ex) lock, semaphore

Synchronization (Critical Section Example)



Synchronization APIs (See also p.67-68)

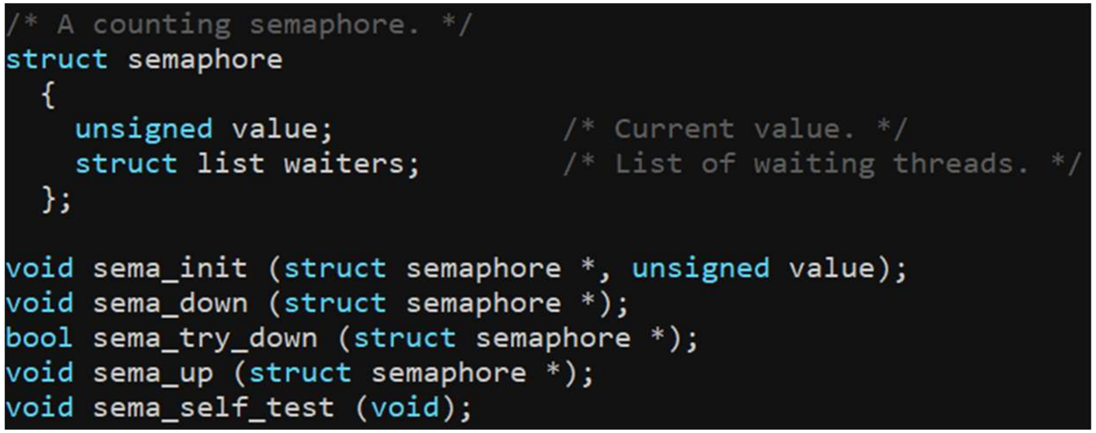
- Lock



```
/* 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 *);
```

- Semaphore



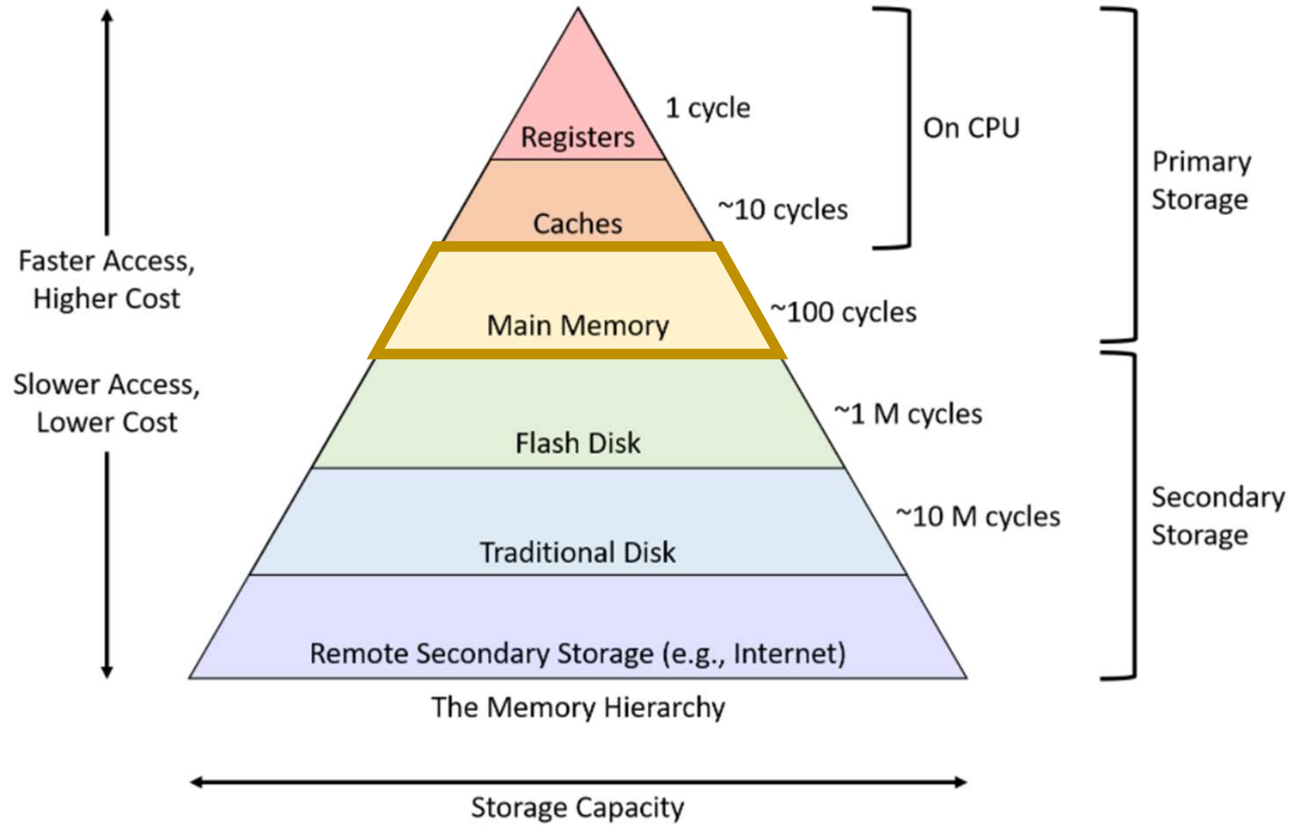
```
/* A counting semaphore. */
struct semaphore
  {
    unsigned value;             /* Current value. */
    struct list waiters;        /* List of waiting threads. */
  };

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);
```

Suggested Order of Implementation

1. Extensible files : Handling with file growth test (growth in length .etc)
2. Subdirectories : multi level of directories
3. Synchoronization : Handling lock issues as usual
4. Buffer cache : Implement the buffer cache and integrate it into the existing filesystem.
5. Rest of test : Remainng mescellaneous items

Buffer Cache

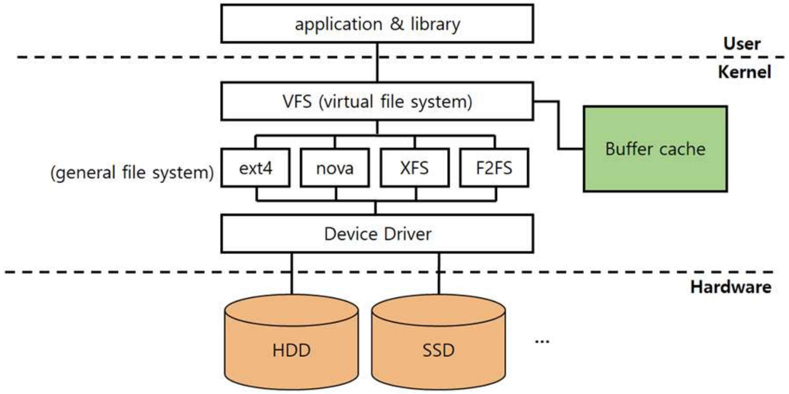


The Memory Hierarchy

Buffer Cache

- **Reading from and writing to disk every IO is inefficient.**
- When a request is made to read or write a block, check to see if it is in the buffer cache.
- If it is in cache, use the cached data without going to disk.
- Otherwise fetch the block from disk into the cache
- Use **eviction algorithm : LRU, clock algorithm**
- cache size : 64 sector size (sector size is 512B, cache size is 512B * 64 = 32KB)

recommendation : integrating the cache into your design early.



page cache vs. buffer cache

- https://naeti.tistory.com/91 (detail)
- https://dhkoo.github.io/2019/03/08/cache/ (simple)

Buffer Cache

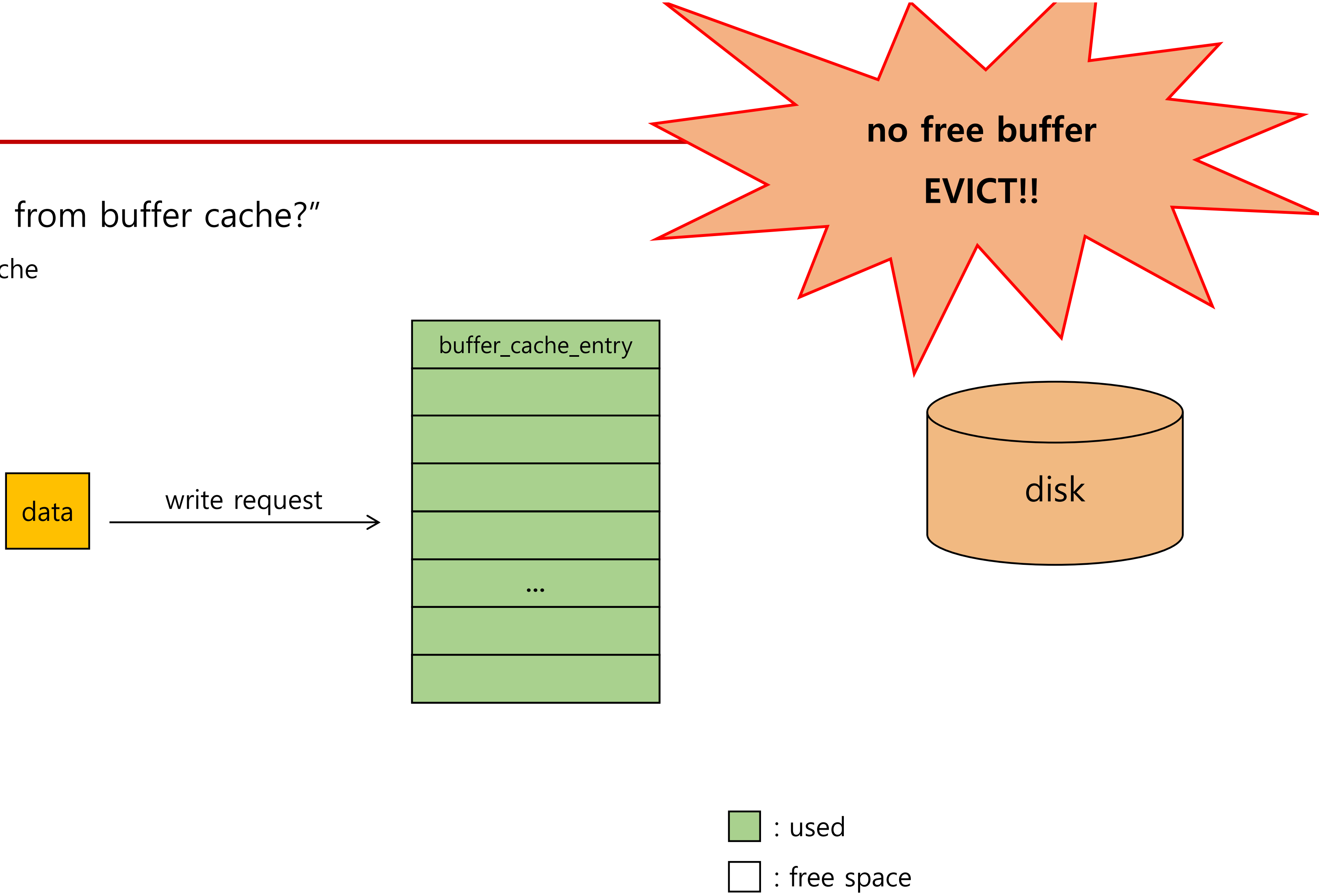
“When to evict the data from buffer cache?”

1. No free space in buffer cache
2. umount or os terminated

Buffer Cache

“When to evict the data from buffer cache?”

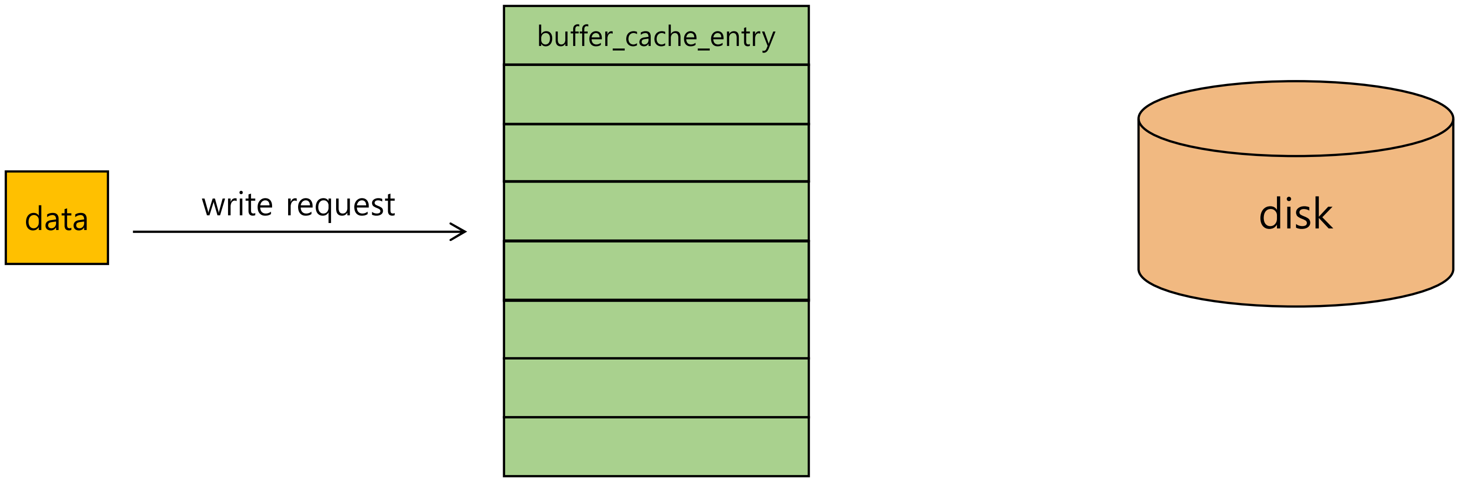
1. No free space in buffer cache
2. umount or os terminated



Buffer Cache

"When to evict the data from buffer cache?"

1. No free space in buffer cache
2. umount or os terminated



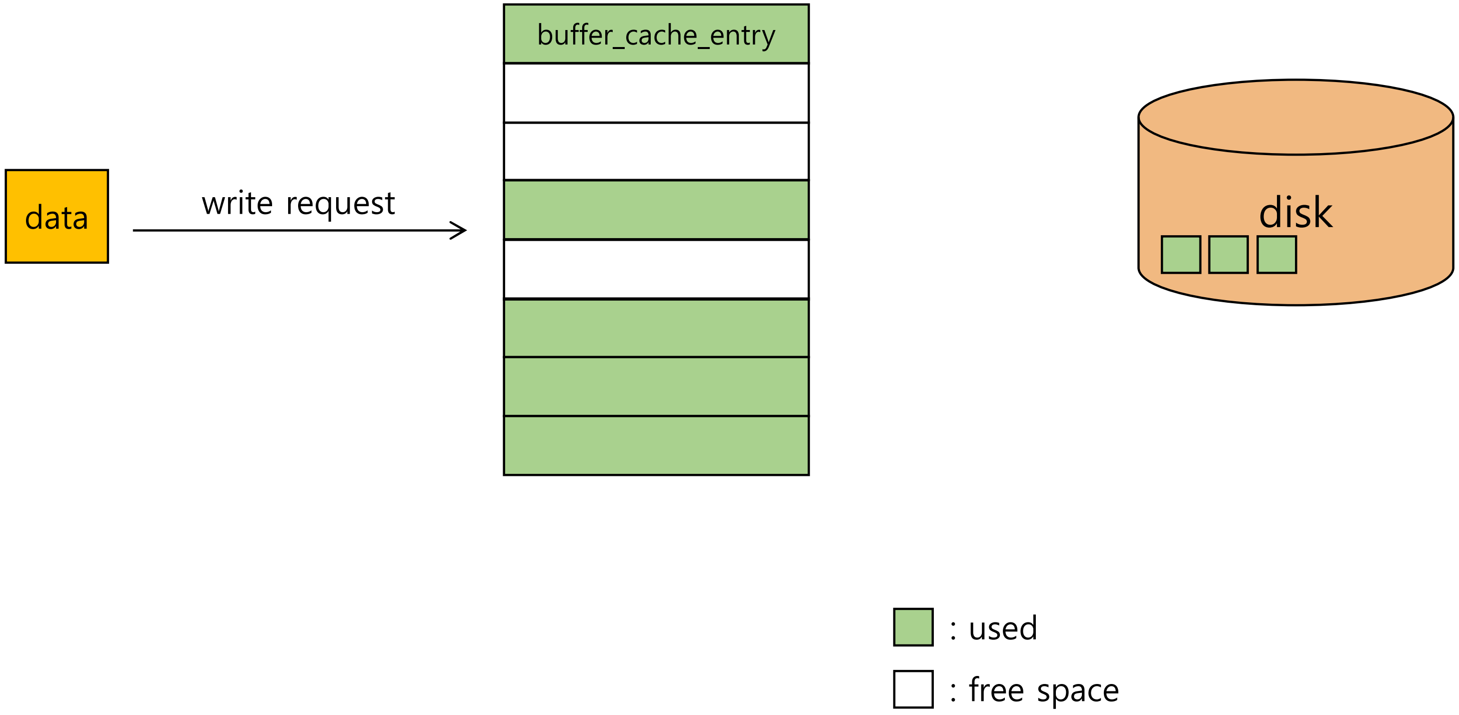
🟩 : used

□ : free space

Buffer Cache

"When to evict the data from buffer cache?"

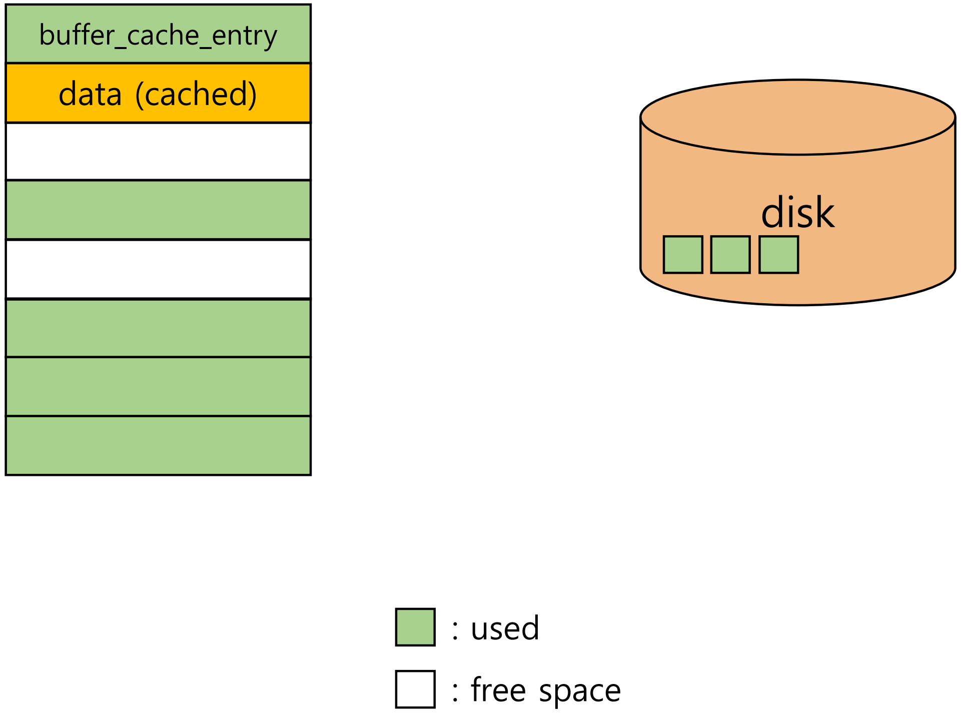
1. No free space in buffer cache
2. umount or os terminated



Buffer Cache

"When to evict the data from buffer cache?"

1. No free space in buffer cache
2. umount or os terminated

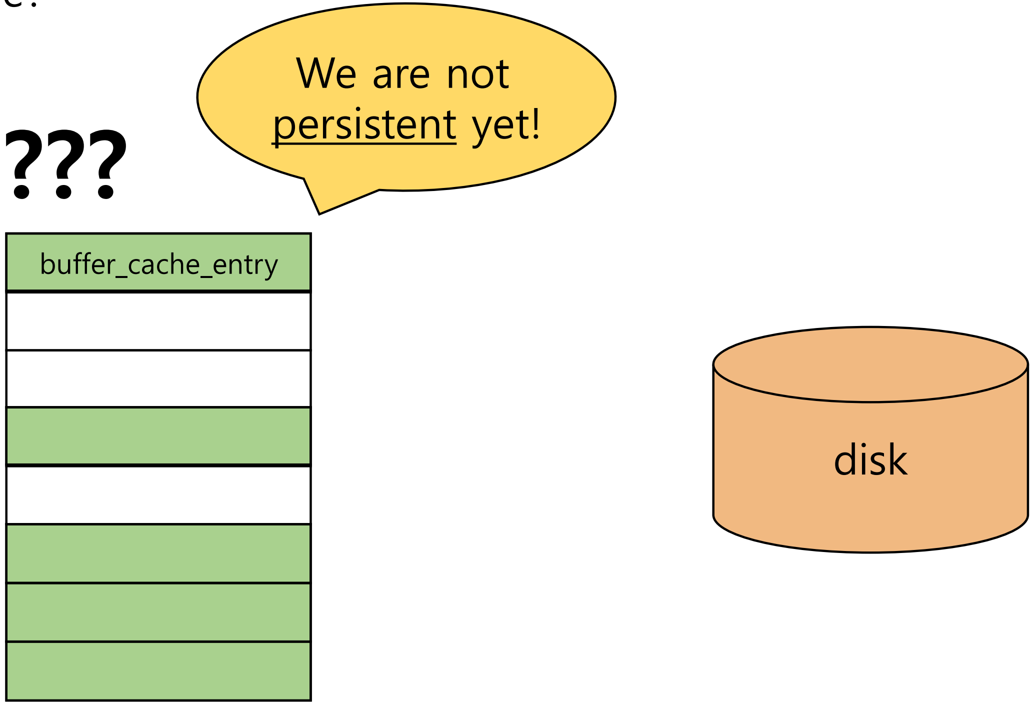


Buffer Cache

"When to evict the data from buffer cache?"

1. No free space in buffer cache
2. umount or os terminated

"filesystem umount or terminated"

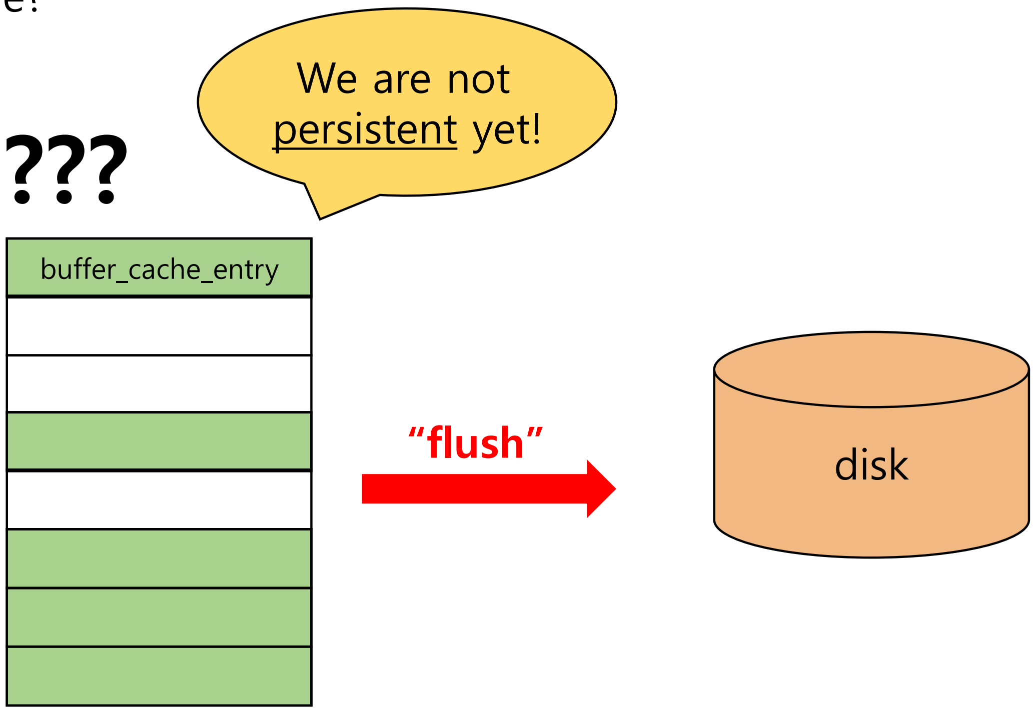


Buffer Cache

"When to evict the data from buffer cache?"

1. No free space in buffer cache
2. umount or os terminated

"filesystem umount or terminated"

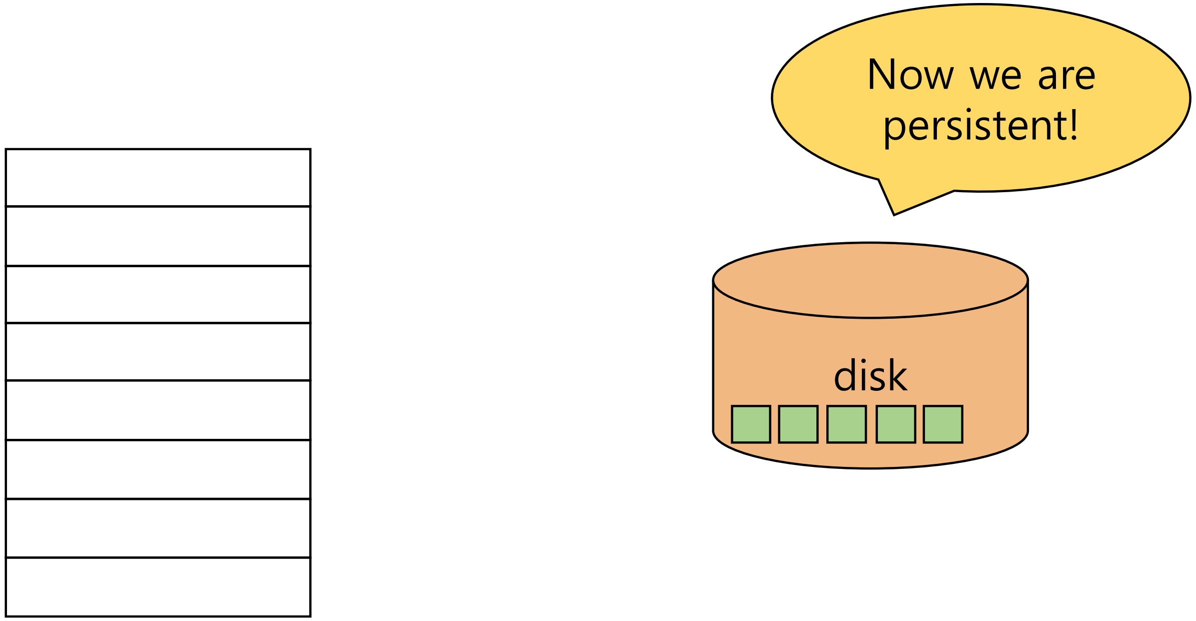


Buffer Cache

"When to evict the data from buffer cache?"

1. No free space in buffer cache
2. umount or os terminated

"filesystem umount or terminated"



Buffer Cache

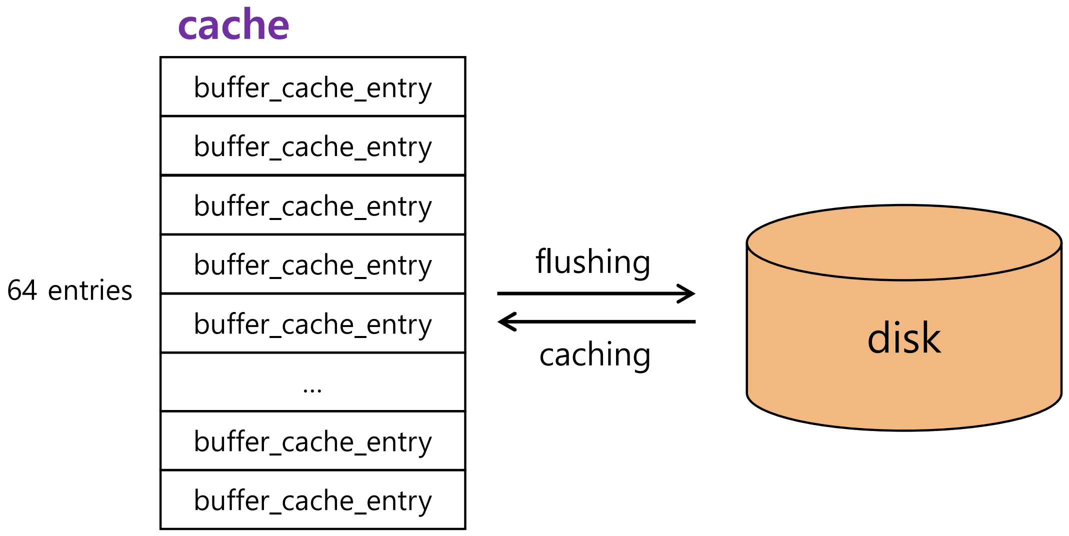
- file to be modified
  - pintos/src/filesys/inode.c
- files to be created
  - pintos/src/filesys/cache.h (you can choose code file name, cache.h, buffer_cache.h ~)
  - pintos/src/filesys/cache.c

- data structure
  - Maximum 64 blocks (64 * 512B = 32KB)

- functions
  - cache search
  - select victim (LRU, clock algorithm)
  - flush
  - write behind
    - When the filesystem is terminated, all dirty blocks should be flushed to the disk
    - When victim entry is selected, dirty block should be flushed to the disk

Buffer Cache (pintos/src/filesys/buffer_cache.c)

```
struct buffer_cache_entry{
        bool valid_bit;
        bool reference_bit; // for clock algorithm
        bool dirty_bit;
        block_sector_t disk_sector;
        uint8_t buffer[BLOCK_SECTOR_SIZE]; //512 * 1B
}
#define NUM_CACHE 64
static struct buffer_cache_entry cache[NUM_CACHE]
static struct lock buffer_cache_lock
…
```



cache

64 entries

buffer_cache_entry

buffer_cache_entry

buffer_cache_entry

buffer_cache_entry

buffer_cache_entry

…

buffer_cache_entry

buffer_cache_entry

flushing

caching

disk

LRU, clock algorithm reference : https://yoon1fe.tistory.com/114

Buffer Cache (function example → src/filesys/buffer_cache.h)

- void buffer_cache_init ();
- void buffer_cache_terminate ();
- void buffer_cache_read (block_sector_t, …);
- void buffer_cache_write (block_sector_t, …);
- struct buffer_cache_entry *buffer_cache_lookup (block_sector_t);
- struct buffer_cache_entry *buffer_cache_select_victim ();
- void buffer_cache_flush_entry(struct buffer_cache_entry*);
- void buffer_cache_flush_all();

Evaluation and Submission

Evaluation

1. 35 tests for project 5 will be graded.
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/filesys
4. Refer to “grade” and “results” files in src/filesys/build after grading
   - “grade” file is only created when you use `make grade`
5. Test cases are classified in extend file system (filesys extend)

Evaluation: Test Cases (35 tests)

filesys/extend		
No.	Name	Point
1	dir-empty-name	3
2	dir-mk-tree	3
3	dir-mkdir	3
4	dir-open	3
5	dir-over-file	3
6	dir-rm-cwd	3
7	dir-rm-parent	3
8	dir-rm-root	3
9	dir-rm-tree	3
10	dir-rmdir	3
11	dir-under-file	3
12	dir-vine	3

filesys/extend		
No.	Name	Point
13	grow-create	3
14	gow-dir-lg	3
15	grow-file-size	3
16	grow-root-lg	3
17	grow-root-sm	3
18	grow-seq-lg	3
19	grow-seq-sm	3
20	grow-sparse	3
21	grow-tell	3
22	grow-two-files	3
23	syn-rw	3
24	dir-empty-name-persistence	3

filesys/extend		
No.	Name	Point
25	dir-mk-tree-persistence	3
26	dir-mkdir-persistence	3
27	dir-open-persistence	3
28	dir-over-file-persistence	3
29	dir-rm-cwd-persistence	3
30	dir-rm-parent-persistence	3
31	dir-rm-root-persistence	3
32	dir-rm-tree-ersistence	3
33	dir-rmdir-persistence	3
34	dir-under-file-persistence	3
35	dir-vine-persistence	3

Evaluation

- Total score:

  filesys(extend) score / 105 x 80

  (max score of filesys(extend) : 3 x 35 = 105)

- Documentation
  - Use the document file uploaded on e-class.
  - Documentation accounts for 20% of total score.

    (Development 80%, Documentation 20%)

Submission

- Make 'ID' directory and copy 'src' directory in the pintos directory and the document file ([ID].docx).
- Compress 'ID' directory to 'os_prj5_[ID].tar.gz'.
- We provide the script 'submit.sh' to make tar.gz file which contains 'src' directory and document file.
  학생들의 편의를 위해 pintos 디렉토리 내 submit.sh 스크립트를 제공합니다.
  이 스크립트는 src 디렉토리와 document file을 포함한 tar.gz 파일을 생성합니다.
- Disclaimer
  - **Any result produced from the 'submit.sh' script is at your own risk.**
  - **You must check the contents of the tar.gz file before submission.**
  - **'submit.sh' 스크립트로 생성된 결과의 모든 책임은 사용자에게 귀속됩니다.**
  - **제출하기 전, tar.gz 파일의 내용물을 반드시 다시 한 번 체크하기 바랍니다.**

Submission

- **Notice – 'submit.sh'**
  - The 'submit.sh' script should be executed on a directory where 'src' folder is located.
    submit.sh 스크립트는 src 폴더가 위치한 디렉토리에서 실행되어야 합니다.
  - 'ID' folder should not be in the directory.
    해당 디렉토리에 '학번' 폴더가 없어야 합니다.
  - 'ID.docx' file should be located in the directory.
    Also, report file with extensions other than 'docx' will not be compressed.
    해당 디렉토리에 '학번.docx' 파일이 있어야 함께 압축됩니다.
    또한 'docx' 이외의 확장자를 가진 보고서 파일은 압축되지 않습니다.
  - Be sure to backup your code in case of an unexpected situation.
    만일의 경우를 대비해 반드시 코드를 백업하여 주세요.

Submission

- It is a **personal project**.
- Due date : 2020. 12. 22 23:59
- Submission
  - The form of submission file is as follows:

Name of compressed file	Example (ID: 20189999)
os_prj5_[ID].tar.gz	os_prj5_20189999.tar.gz

  - No hardcopy.
  - **Copy will get a penalty (1st time: 0 Point and downgrading, 2nd time: F grade)**

Submission

- **Contents**
  - ① Pintos source codes (Only **'src' directory** in pintos directory)
    최소한의 용량을 위해 **src 디렉토리만** 압축파일에 포함합니다.
  - ② Document: **[ID].docx** (e.g. 20189999.docx; Other format is not allowed such as .hwp)

- **How to submit**
  1) Make tar.gz file.
     - Copy the document file ([ID].docx) to pintos directory.
     - **Execute submit.sh script in the pintos directory and follow the instructions of the script**.
       pintos 디렉토리 내의 submit.sh 스크립트를 실행하고 스크립트의 지시를 따르십시오.
     - Check that **os_prj5_[ID].tar.gz** is created.
     - Decompress **os_prj5_[ID].tar.gz** and check the contents in it. ($ **tar -zxf os_prj5_[ID].tar.gz**)
       (Only **[ID].docx** and **src** directory should be contained in the tar.gz file.)
     - For example, if your ID is 20189999, os_prj5_20189999.tar.gz should be created.
       To decompress the tar.gz file, execute **tar -zxf os_prj5_20189999.tar.gz**
     - **Please check the contents of tar.gz file after creating it.**
  2) Upload the **os_prj5_[ID].tar.gz** file to e-class.

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