Operating System Lab
Part 4: Filesystem

KAIST EE

Youjip Won

Indexed and Extensible File

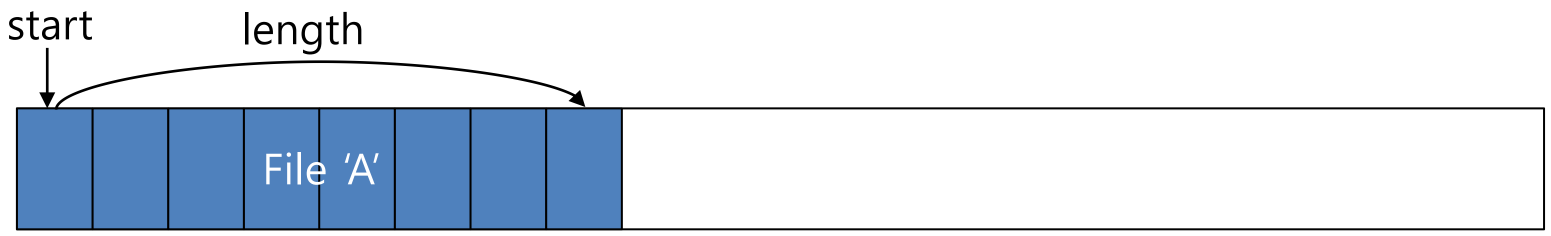
Extensible File

- Goal
  - In original pintos, file size is fixed when it is created.
  - In this project, we will modify pintos filesystem to change the file size dyn amically. Maximum file size will be 8MB.

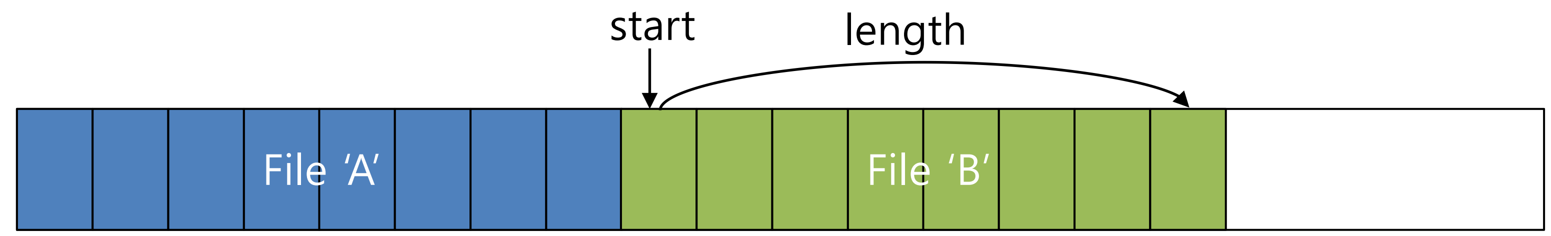
- Files to modify
  - pintos/src/filesys/inode.c
  - pintos/src/filesys/inode.h

How to allocate block when pintos create file

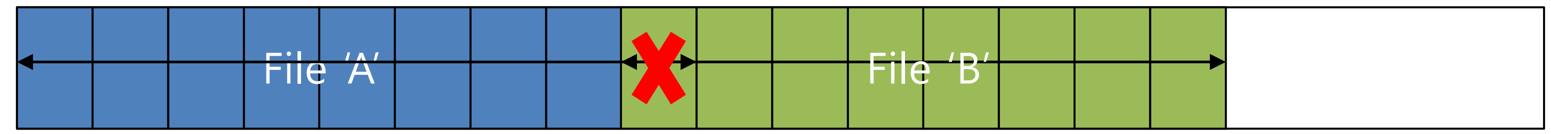
1. Creation of file A (Save start block address and length to inode)



2. Creation of file B (Save start block address and length to inode)

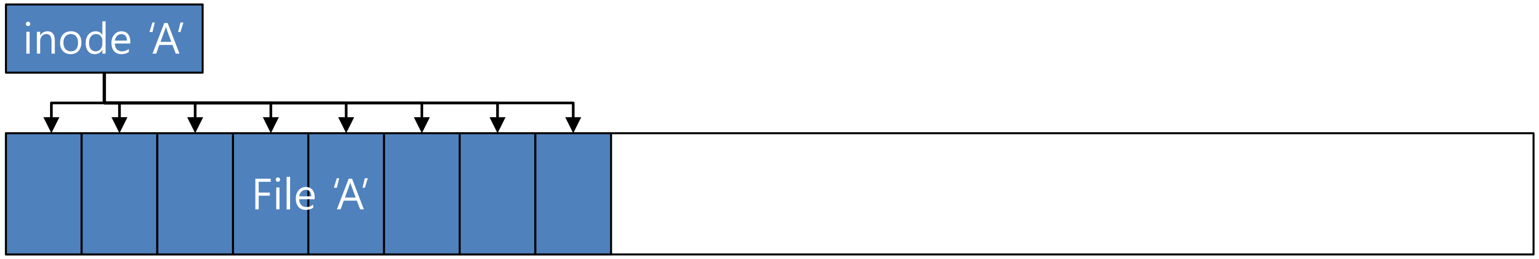


3. The size of file A can't be increased because of conflict with file B

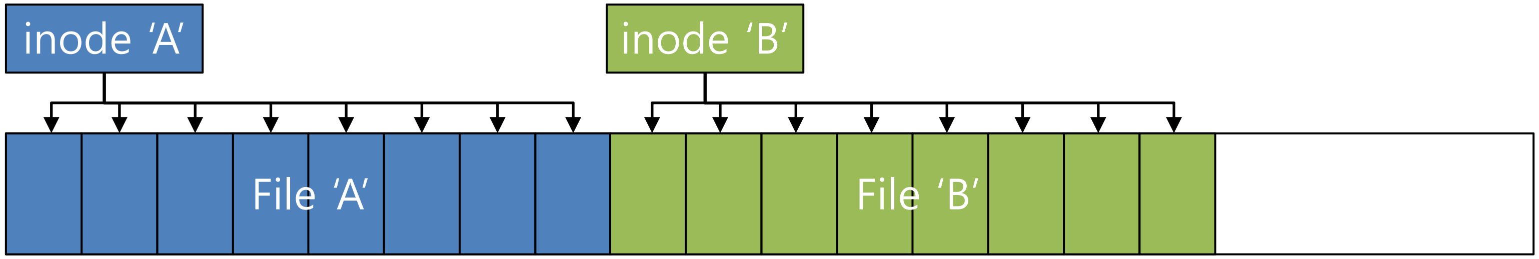


After modification

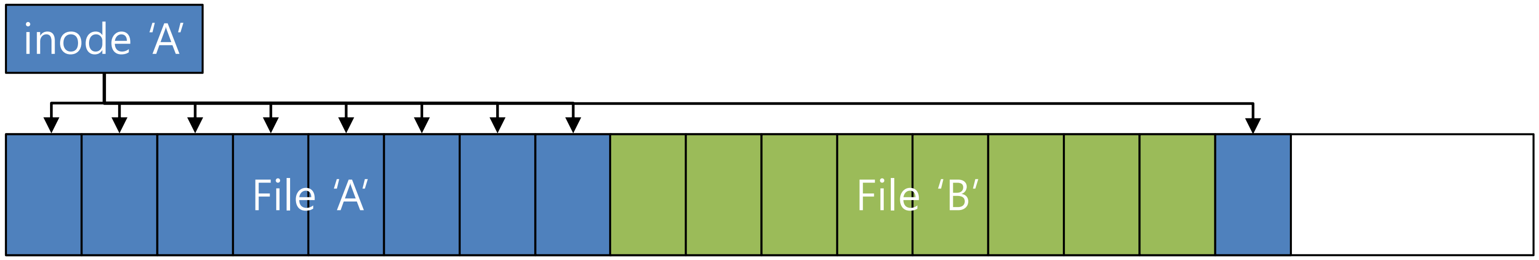
1. Creation of file A (Save block addresses of all each blocks to inode)



2. Creation of file B (Save block addresses of all each blocks to inode)

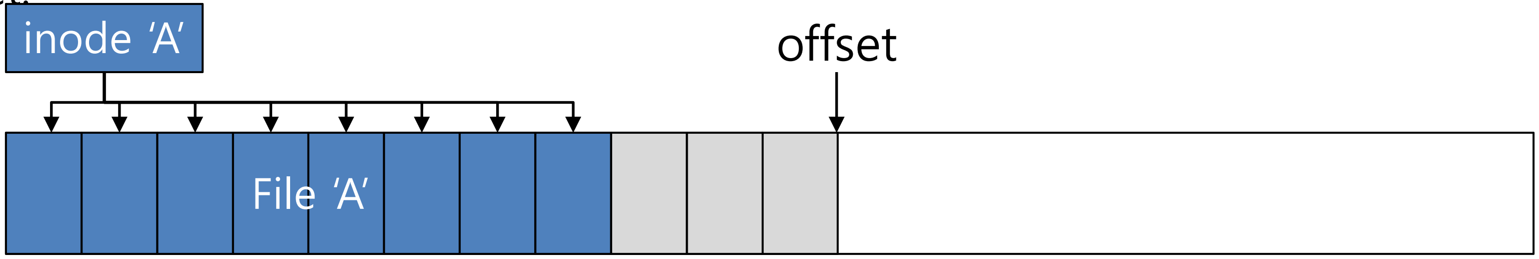


3. Expanding the file A can be done by allocating a new block and by saving its address to inode of file A



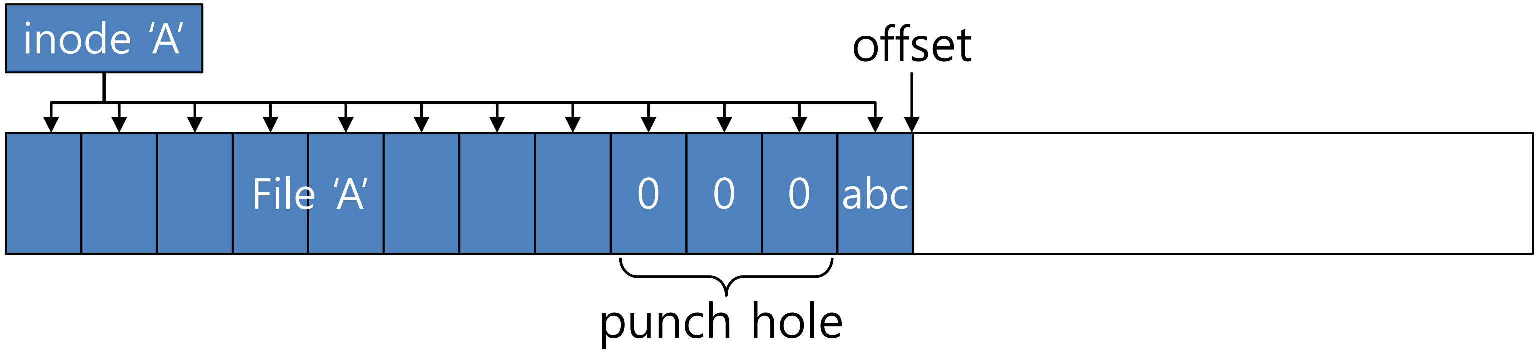
Some details

1. The seek() does not change the file size nor does it allocate the blocks. It just updates offset.



2. When a `write()` is called and offset is larger than the file size, the file size is updated and blocks are allocated (Fill punch hole with zero).

After write ('A', "abc", size):



To Do's

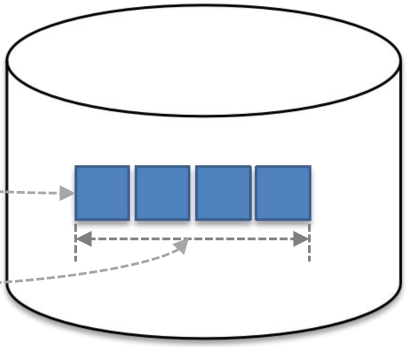
1. Modify on-disk inode structure (`struct inode_disk`).
2. Modify code using on-disk inode.
   - Changing file offset to block address
     - `static block_sector_t byte_to_sector(const struct inode_disk *inode_disk, off_t pos)`
   - Creating new inode
     - `bool inode_create(block_sector_t sector, off_t length)`
   - Deleting an inode
     - `void inode_close(struct inode *inode)`
3. Handle extension of file.
   - `off_t inode_write_at(struct inode *inode, const void *buffer_, off_t size, off_t offset)`

On-disk inode in current pintos

- Fields for pointing blocks in current pintos
  - `start`: Start block address
  - `length` : Size of file (byte)
    - Fixed when the file is created.
    - All blocks should be continuous.

pintos/src/filesys/inode.c

```
struct inode_disk
{
    block_sector_t start;   /* First data sector */
    off_t length;           /* File size in bytes */
    unsigned magic;         /* Magic number */
    uint32_t unused[125];   /* Not Used */
}
```

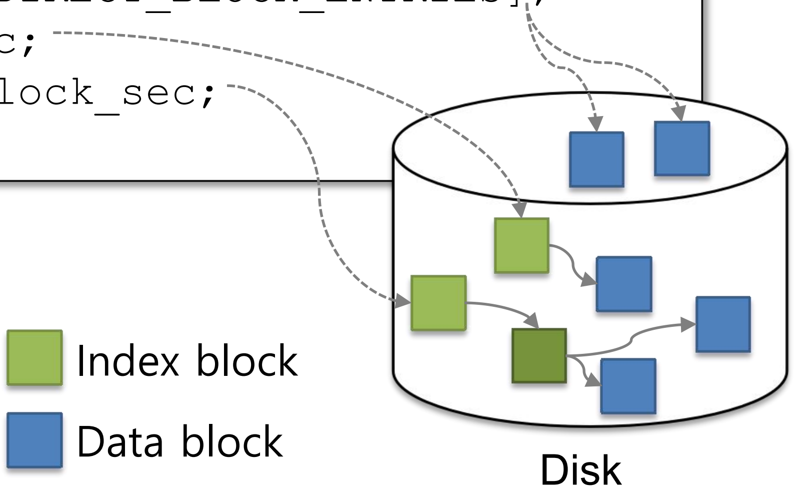

Disk

To Do 1 - Modify on-disk inode structure

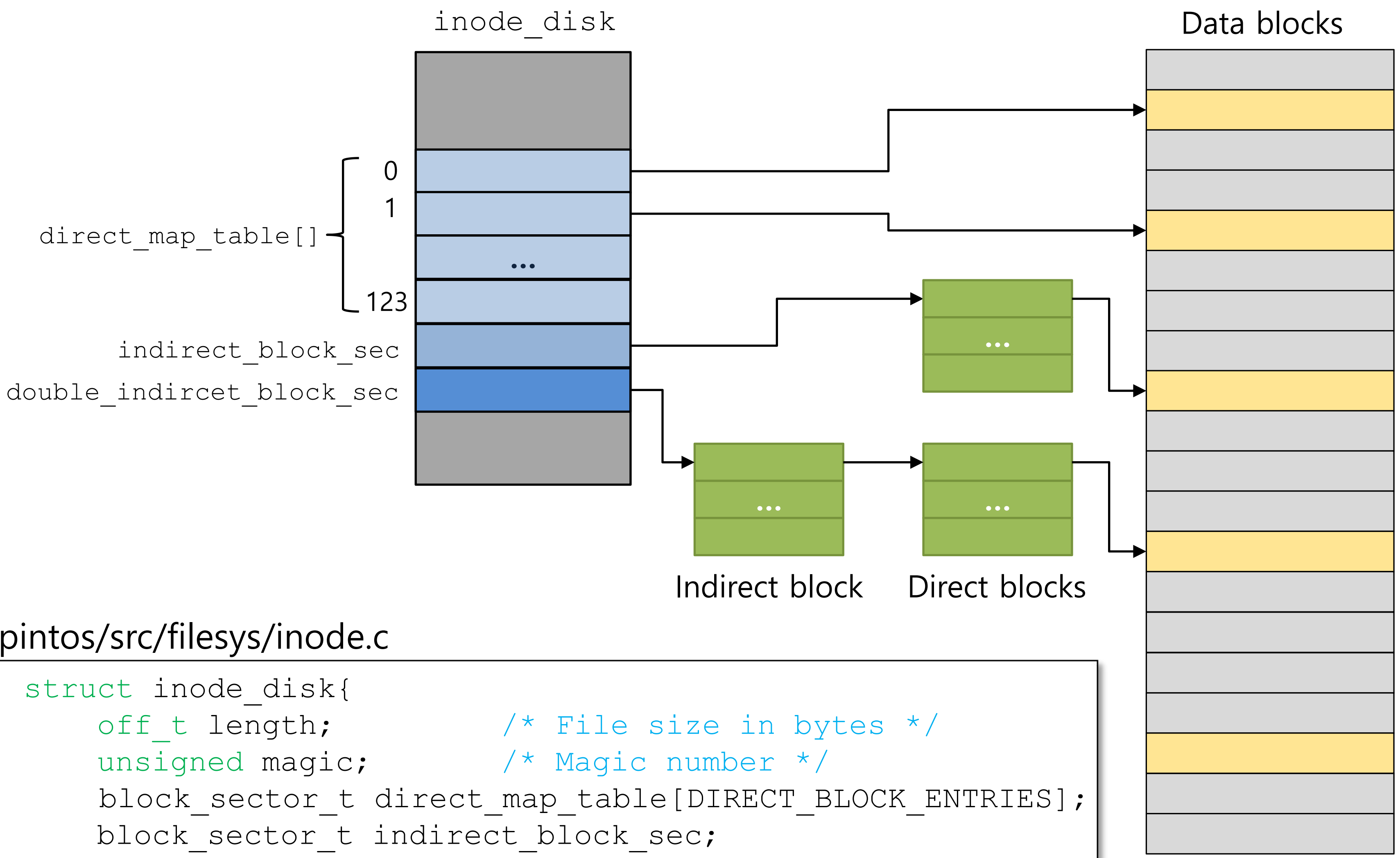
- Modify `struct inode_disk.`
  - Add block pointers of direct, indirect, double indirect.
  - We must maintain the size of "`struct inode_disk`" by adjusting the nu
    mber of direct block pointers.

pintos/src/filesys/inode.c

```
struct inode_disk{
    off_t length;              /* File size in bytes */
    unsigned magic;            /* Magic number */
    block_sector_t direct_map_table[DIRECT_BLOCK_ENTRIES];
    block_sector_t indirect_block_sec;
    block_sector_t double_indirect_block_sec;
}
```



On-disk inode with block indexing



pintos/src/filesys/inode.c

```
 struct inode_disk{
      off_t length;            /* File size in bytes */
      unsigned magic;          /* Magic number */
      block_sector_t direct_map_table[DIRECT_BLOCK_ENTRIES];
      block_sector_t indirect_block_sec;
      block_sector_t double_indirect_block_sec;
 }
```

To Do 2 – compute the sector number from the file offset

- `static block_sector_t byte_to_sector(const struct inode_disk *inode_disk, off_t pos)`
  - Return block address associated an offset of inode.

- In original: just return sum of start and pos.
- Modify code in red box to use block indexing.

pintos/src/filesys/inode.c

```
static block_sector_t byte_to_sector (const struct inode *inode, off_t pos){
  ASSERT (inode != NULL);
  if (pos < inode->data.length)
    return inode->data.start + pos / BLOCK_SECTOR_SIZE;
  else
    return -1;
}
```

To Do 2 – Creating an inode

- `bool inode_create(block_sector_t sector, off_t length)`
  - Create file which have size of length.
- In original: Allocate contiguous blocks and save its start address.
- Modify code to save the block addresses of all blocks allocated.

pintos/src/filesys/inode.c

```
bool inode_create (block_sector_t sector, off_t length){
  …
  disk_inode = calloc (1, sizeof *disk_inode);
  if (disk_inode != NULL){
    disk_inode->length = length;
    disk_inode->magic = INODE_MAGIC;
    if (free_map_allocate(sectors, &disk_inode->start)) {
      block_write(fs_device, sector, disk_inode);
      if (sectors > 0) {
        /* Fill file out by zero */
      }
      success = true;
    }
    free(disk_inode);
  }
  return success;
}
```

To Do 2 – Deleting an inode

- When it deletes an inode,
  - We have to deallocate all blocks inode have.
  - Add block deallocating code at inode_close.

pintos/src/filesys/inode.c

```
void inode_close (struct inode *inode){
    …
    /* Release resources if this was the last opener. */
    if (--inode->open_cnt == 0){
        …
        /* Deallocate blocks if removed. */
        if (inode->removed){
            /* Get on-disk inode structure by get_disk_inode()*/
            /* Deallocate each blocks by free_inode_sectors() */
            /* Deallocate on-disk inode by free_map_release() */
        }
  …
```

Remove original code and add new code

To Do 3 – Handle extension of file

- When the file size changes,
  - Allocate a new block and update data block pointer in inode.
  - Fill the allocated blocks with zero.

pintos/src/filesys/inode.c

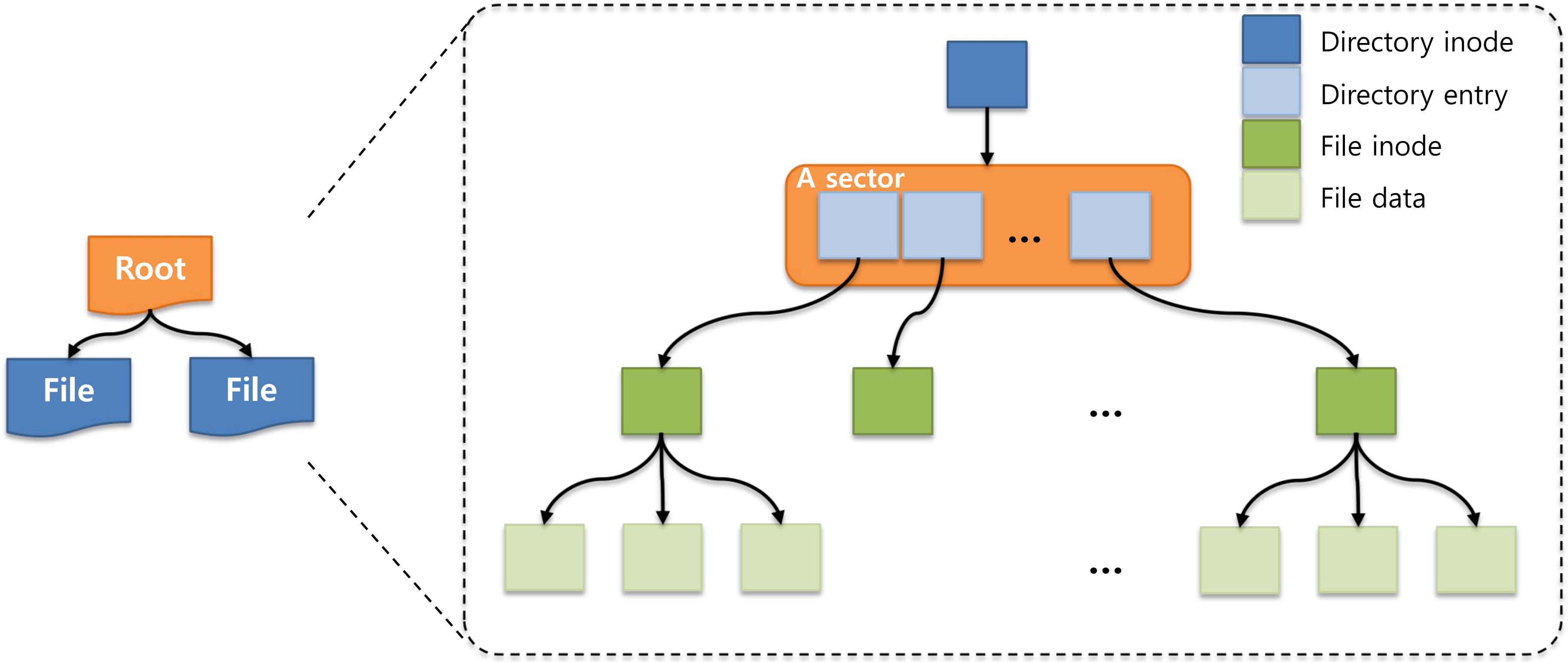
```
off_t inode_write_at (struct inode *inode, const void *buffer_,
                      off_t size, off_t offset) {
    …
    /* Acquire some lock to avoid contention on inode */

    int old_length = disk_inode->length;
    int write_end = offset +  size - 1;

    if(write_end > old_length -1 ){
        /* When size of file is updated, Update inode */
    }
    /* Release lock */

    while(size > 0){
    …
```

Subdirectory

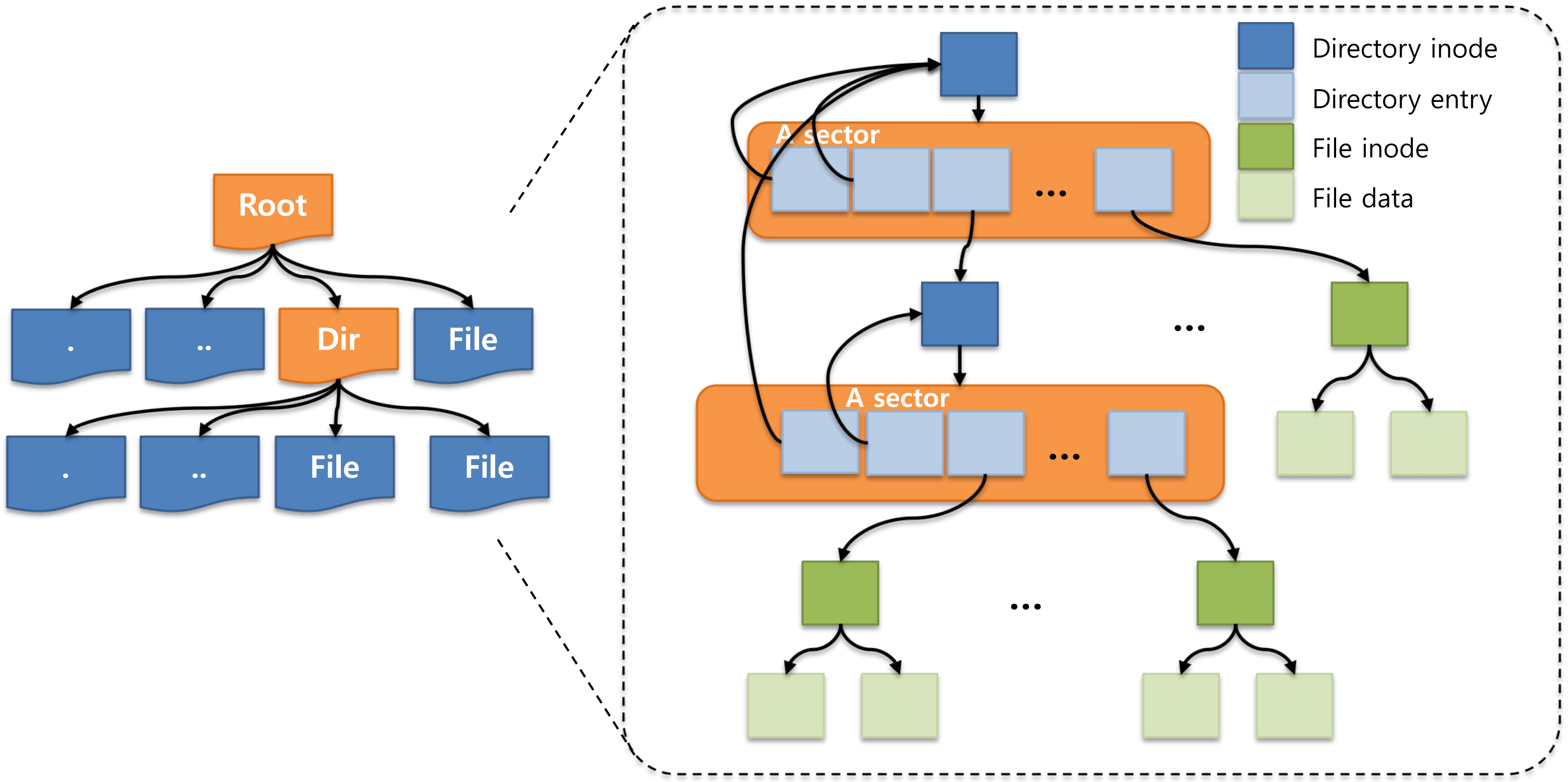
Overview of Subdirectories

- Main Goal
  - Original pintos has only root directory but not the other subdirectories.
  - We will implement subdirectory feature to make filesystem of pintos to ha ve hierarchical structure.
- Files to modify
  - pintos/src/filesys/inode.c
  - pintos/src/filesys/filesys.*
  - pintos/src/filesys/file.*
  - pintos/src/filesys/directory.*
  - pintos/src/userprog/syscall.c

Directory structure in original pintos



Hierarchical directory structure



Requirements

- Implement Hierarchical directory structure.
  - Make directory entry can point to not only regular file but also directory.
  - Add directory entry '.' and '..'.
- Implement "current directory" for a thread.
  - Distinguish absolute path and relative path (Distinguisher is '/').
- Modify directory related functions.
  - `filesys_create(), filesys_open(), filesys_remove()`
- Add new directory related system calls.

Now, **there is a concept of path** !!!

To Do's

1. Add flag to indicate whether the inode is for regular file or for directory.
2. Define current directory pointer in `struct thread`.
3. Modify code for directory manipulation.
   1. Creating new file.
   2. Opening a file.
   3. Removing a file.
4. Add system calls for directory manipulation.
5. Add special directory entries.: ".", "..".

To Do 1: Modify On-disk inode

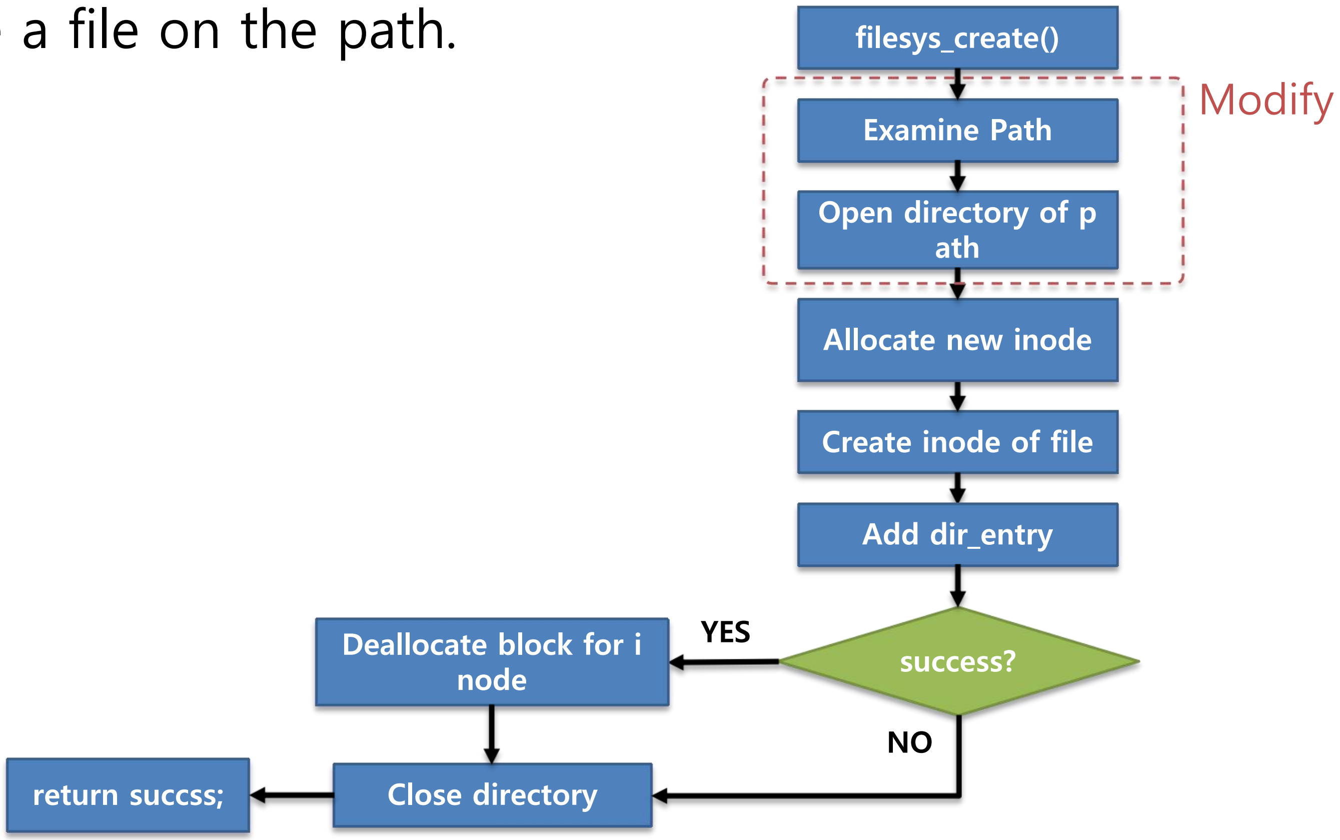
- Add a flag indicating if it is file or directory.
  - `struct inode_disk`
    - file: pintos/src/filesys/directory.c
    - `Regular file(=0), directory(=1)`
- When a file is created, we should set this flag properly.

To Do 2: Define current directory in `struct thread`.

- Add current directory pointer in thread.
  - `struct thread`
    - File: pintos/src/threads/thread.h
    - Add pointer to `struct dir` representing current directory.
- When a thread is created, the child thread inherits parent's current directory.

To Do 3: Modify algorithm of file creation

- Distinguish if it is absolute path or relative path.
- Find directory associated with path.
- Create a file on the path.



To Do 3: Modify algorithm of File creation (Cont.)

- `bool filesys_create (const char *name, off_t initial_size)`
  - Original: Always create a file in the root directory.
  - After modification: Parse the path and create that file on that directory.
    - Distinguish absolute path and relative path and parse.
  - Add the code to set `is_dir` flag to 0 if it is regular file.
  - Add new directory entry to directory of path.

To Do 3: Opening a file

- `struct file *filesys_open(const char *name)`
  - Original: Always find the file on the root directory.
  - After modification: Parse path, find the file on that directory, and open it.
    - Distinguish absolute path and relative path and parse.
      - When the path is absolute: Find from the root directory.
      - When the path is relative: Find from the current directory.

To Do 5: Removing a file

- `bool filesys_remove(const char *name)`
  - Original: Always remove file from root directory.
  - After modification: Remove file from directory specified by path.
    - Distinguish absolute path and relative path and parse.
  - Check if in-memory of target file is for directory or regular file.
    - If it is directory, check it have files.
      - Remove only when directory is empty.
    - If it is file, just remove it.

To Do 6: Add system calls about directory

- `bool chdir(const char *dir)`
  - Change the current working directory of the process to `dir`.
  - Return true if successful, false on failure.
- `bool mkdir(const char *dir)`
  - Creates the directory named `dir`.
  - Returns true if successful, false on failure.
- `bool readdir(int fd, char *name)`
  - Reads a directory entry from file descriptor `fd`, which must represent a directory.
  - If successful, stores file name in name and return true.
    - `'.'` and `'..'` should not be returned by `readdir`.
- `bool isdir(int fd)`
  - Returns true if `fd` represents a directory, false if it represents an ordinary file.
- `int inumber(int fd)`
  - Returns the inode number of the inode associated with `fd`.

To Do 7: Add special entries

- Special directory entries.
  - `'.'`: it represents itself.
  - `'..'`: it represents its parent directory.
- When a directory is created, special entries should be added.
  - even root directory created during format.
- If the user tries to remove them, system call should return fails.