Operating System Lab
Part 4: Filesystem

KAIST EE

Youjip Won

Filesystem: Background

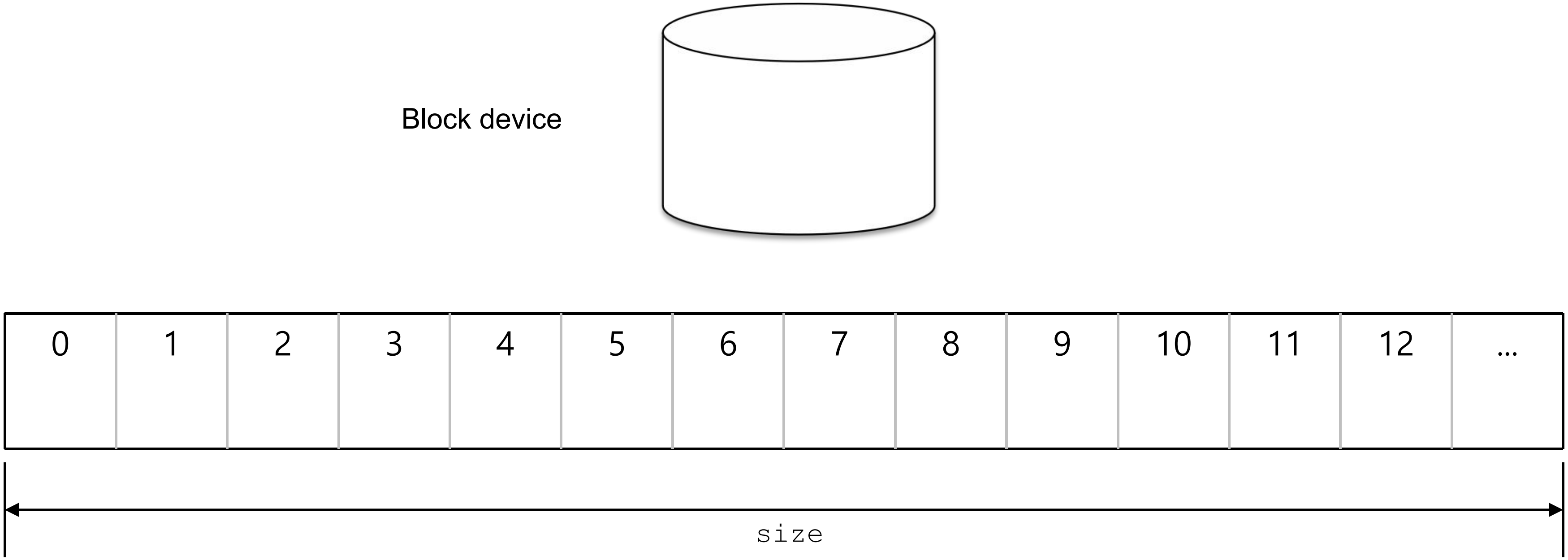
Block Device

0	1	2	3	4	5	6	7	8	9	10	11	12	…

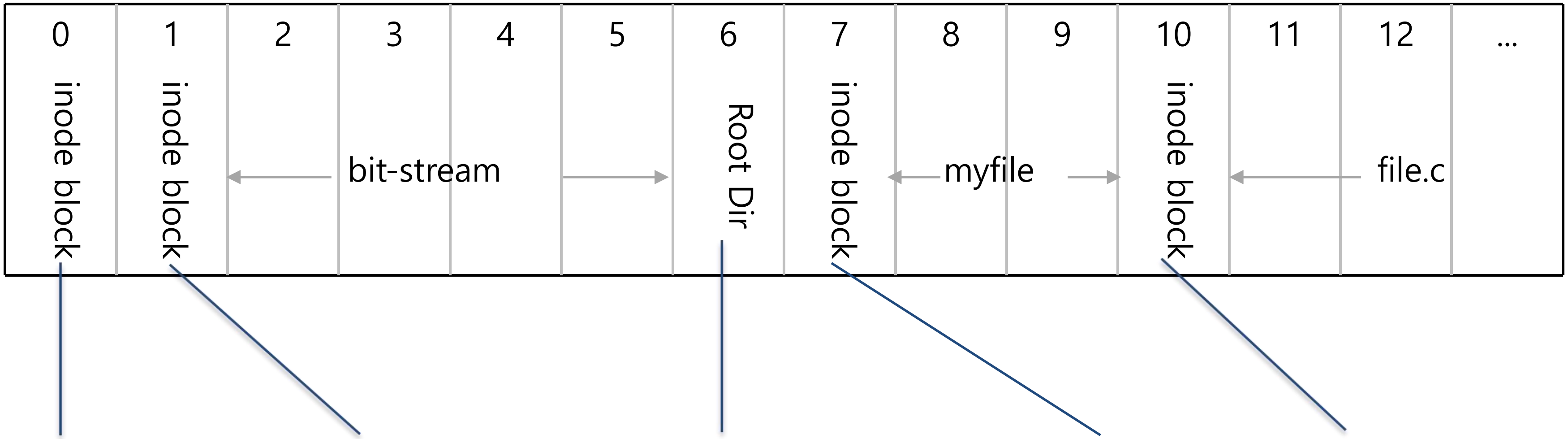
Logical block address (sector number)

Abstraction for Block Device

```
struct block  {
          struct list_elem list_elem;          /* Element in all_blocks. */
          char name[16];                       /* Block device name. */
          enum block_type type;                 /* Type of block device. */
          block_sector_t size;                  /* Size in sectors. */
          const struct block_operations *ops;   /* Driver operations. */
          void *aux;                           /* Extra data owned by driver. */
          unsigned long long read_cnt;         /* Number of sectors read. */
          unsigned long long write_cnt;        /* Number of sectors written. */
};
```



Filesystem layout



Inode for bitmap

```
start : 2
length : 204
8
magic
unused[125]
```

Inode for "/"

```
start : 6
length : 320
magic
unused[125]
```

entries in "/"

fie name	inode No.
myfile	7
file.c	10
...	

myfile inode

```
start : 8
length : 1024
magic
unused[125]
```

file.c inode

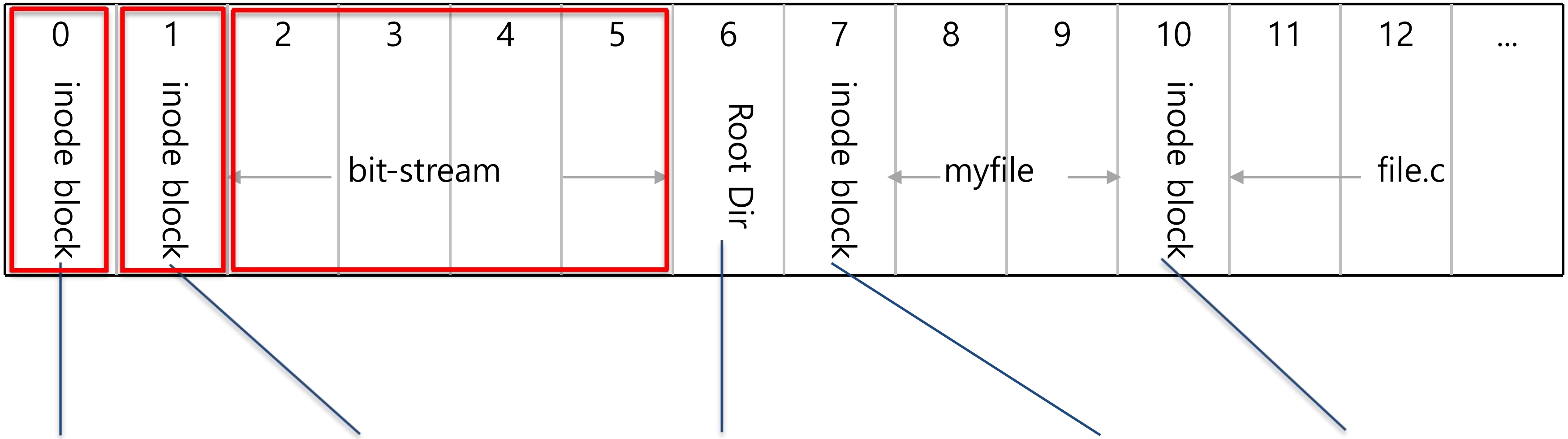
```
start : 11
length : 204
8
magic
unused[125]
```

Bitmap size:
2048 bytes (4 sectors) = 2048*8 bits = 16384 bits
Bitmap for 16384 sectors (8 Mbyte) = 16384 * 512 Byte
Bitmap for 8 Mbyte filesystem partition

Formatting a filesystem in Pintos

- Create a filesystem layout on disk.
  - Create and initialize bitmap.
  - Create inode of bitmap and write its data on the disk.
  - Create inode of Root directory.

Formatting a filesystem in Pintos



Inode for bitmap

```
start : 2
length : 204
8
magic
unused[125]
```

Inode for "/"

```
start : 6
length : 320
magic
unused[125]
```

entries in "/"

fie name	inode No.
myfile	7
file.c	10
...	

myfile inode

```
start : 8
length : 1024
magic
unused[125]
```

file.c inode

```
start : 11
length : 204
8
magic
unused[125]
```

Bitmap size:
2048 bytes (4 sectors) = 2048*8 bits = 16384 bits
Bitmap for 16384 sectors = 16384 * 512 Byte
Bitmap for 8 Mbyte filesystem partition

Formatting a filesystem in Pintos

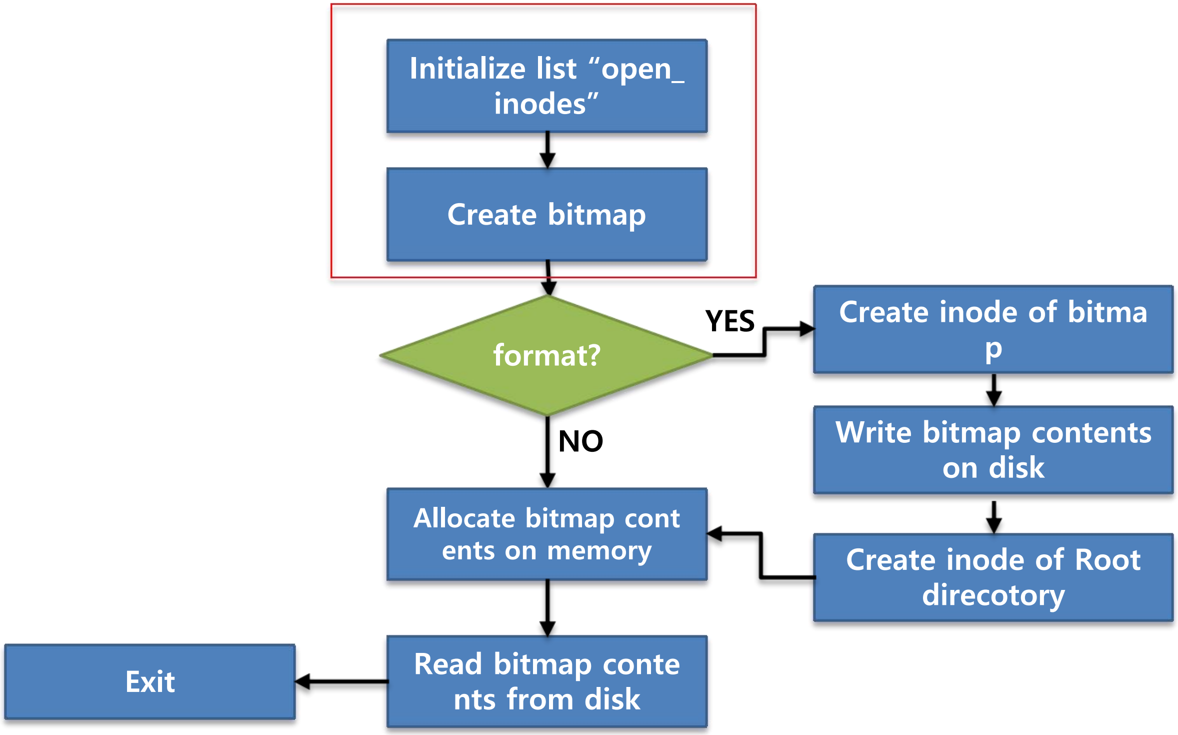
pintos/src/filesys/filesys.c

```
void filesys_init (bool format) {
     fs_device = block_get_role (BLOCK_FILESYS);

     inode_init ();
     free_map_init ();
     if (format)
          do_format ();
     free_map_open ();
}
```

- Create and write bitmap of Filesystem.
  - `inode_init()`: Create and initialize the data structure for open files.
  - `free_map_init()` : Create and initialize bitmap.
  - `do_format()`
    - Create and write the inode of bitmap file.
    - Create the inode of Root directory.

Initializing the filesystem


Initialize list "open_ inodes"
Create bitmap
format?
YES
NO
Create inode of bitma p
Write bitmap contents on disk
Create inode of Root direcotory
Allocate bitmap cont ents on memory
Read bitmap conte nts from disk
Exit

Initialize the list of open inodes

- Initialize the list of in-memory inodes.
  - open_inodes : global list of in-memory inode (Doubly linked list)
  - **What is the data structure that represents the open files in xv6 ?**

pintos/src/filesys/inode.c

```c
static struct list open_inodes;
void inode_init (void)
{
    list_init (&open_inodes);
}
```

```c
void list_init (struct list *list)
{
    ASSERT (list != NULL);
    list->head.prev = NULL;
    list->head.next = &list->tail;
    list->tail.prev = &list->head;
    list->tail.next = NULL;
}
```

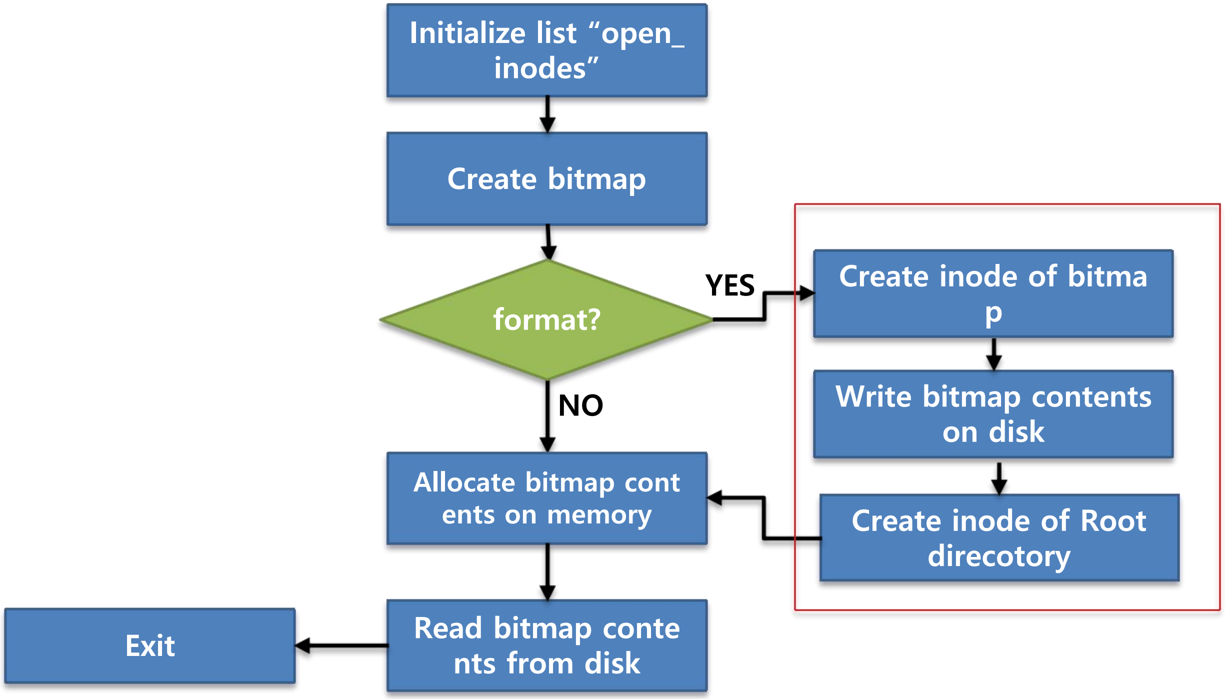
Create and initialize the bitmap

- Create and initialize the bitmap in-memory.
  - `bitmap_create()` : Create bitmap and fill it out 0.
  - `bitmap_mark()`: Flip bit indexed by second parameter.

pintos/src/filesys/free-map.c

```
void free_map_init (void) {
    free_map = bitmap_create (block_size (fs_device));
    …
    /* FREE_MAP_SECTOR = 0, ROOT_DIR_SECTOR = 1*/
    bitmap_mark (free_map, FREE_MAP_SECTOR);
    bitmap_mark (free_map, ROOT_DIR_SECTOR);
}
```

Initializing the filesystem


Initialize list "open_ inodes"
Create bitmap
format?
YES
NO
Create inode of bitma p
Write bitmap contents on disk
Create inode of Root direcotory
Allocate bitmap cont ents on memory
Read bitmap conte nts from disk
Exit

Create the inodes for bitmap and root directory

- Store the bitmap on the disk and Initialize root directory.
  - `free_map_create()`: Create inode of bitmap.
    - Write contents of bitmap on disk.
  - `dir_create()` : Create inode of root directory.
    - The maximum number of files in Root directory is 16.

pintos/src/filesys/filesys.c

```c
static void do_format (void){
    free_map_create ();
    if (!dir_create (ROOT_DIR_SECTOR, 16))
        PANIC ("root directory creation failed");
    free_map_close ();
}
```

Create and save the bitmap file

- Create and write inode of bitmap on disk.
  - `inode_create()` : Create inode of bitmap at block 0.
  - `bitmap_write()` : Write contents of bitmap on disk.

pintos/src/filesys/free_map.c

```
void free_map_create (void) {
    /* Create inode. */
    if (!inode_create (FREE_MAP_SECTOR, bitmap_file_size (free_map)))
        PANIC ("free map creation failed");
    free_map_file = file_open (inode_open (FREE_MAP_SECTOR));
    …
    /* Write bitmap to file. */
    if (!bitmap_write (free_map, free_map_file))
}
```

Create a root directory.

- Create inode of root directory at block 1.
  - sector : block number which has the inode for root directory.
  - entry_cnt : The maximum number of entries in root directory.

pintos/src/filesys/directory.c

```
bool dir_create (block_sector_t sector, size_t entry_cnt)
{
    return inode_create (sector, entry_cnt * sizeof (struct di
r_entry));
}
```

Creating a root directory

- Allocate data blocks for root directory and save its start address at inode.

pintos/src/filesys/inode.c

```
bool inode_create (block_sector_t sector, off_t length){
      …
      size_t sectors = bytes_to_sectors(length);
      if (free_map_allocate (sectors, &disk_inode->start)){
            block_write(fs_device, sector, disk_inode);
      }
      …
      return success;
}
```

- `bytes_to_sectors()`: Translate length of bytes to length of blocks.
- `free_map_allocate()`: Allocate contiguous blocks and save its start address at second parameter.
- `block_write()`: write `disk_inode` to disk.

Synchornizing the bitmap to the disk

- Close inode of bitmap: Deallocate and remove in-memory inode from `open_inodes` list.

pintos/src/filesys/free-map.c

```
void free_map_close (void)
{
    file_close (free_map_file);
}
```

- `file_close()`
  - Remove inode from `open_inodes` list and deallocate it.
  - Deallocate file structure.

Load the bitmap to memory

- Read bitmap contents on disk.

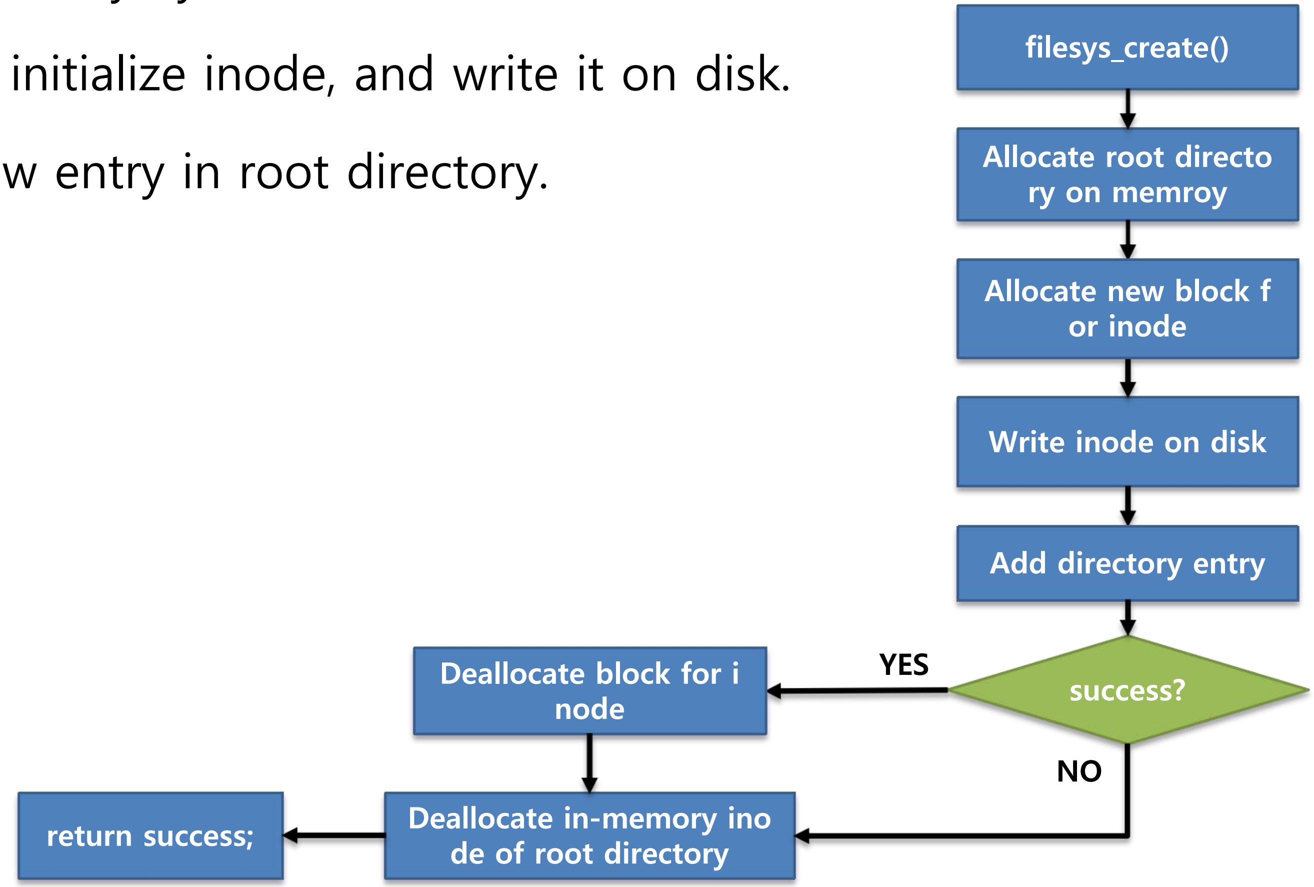
pintos/src/filesys/free-map.c

```c
void free_map_open (void) {
  free_map_file = file_open (inode_open (FREE_MAP_SECTOR));
  if (free_map_file == NULL)
    PANIC ("can't open free map");
  if (!bitmap_read (free_map, free_map_file))
    PANIC ("can't read free map");
}
```

- `file_open()` : Allocate and initialize file structure.
- `bitmap_read()` : Read bitmap contents on disk.

File create

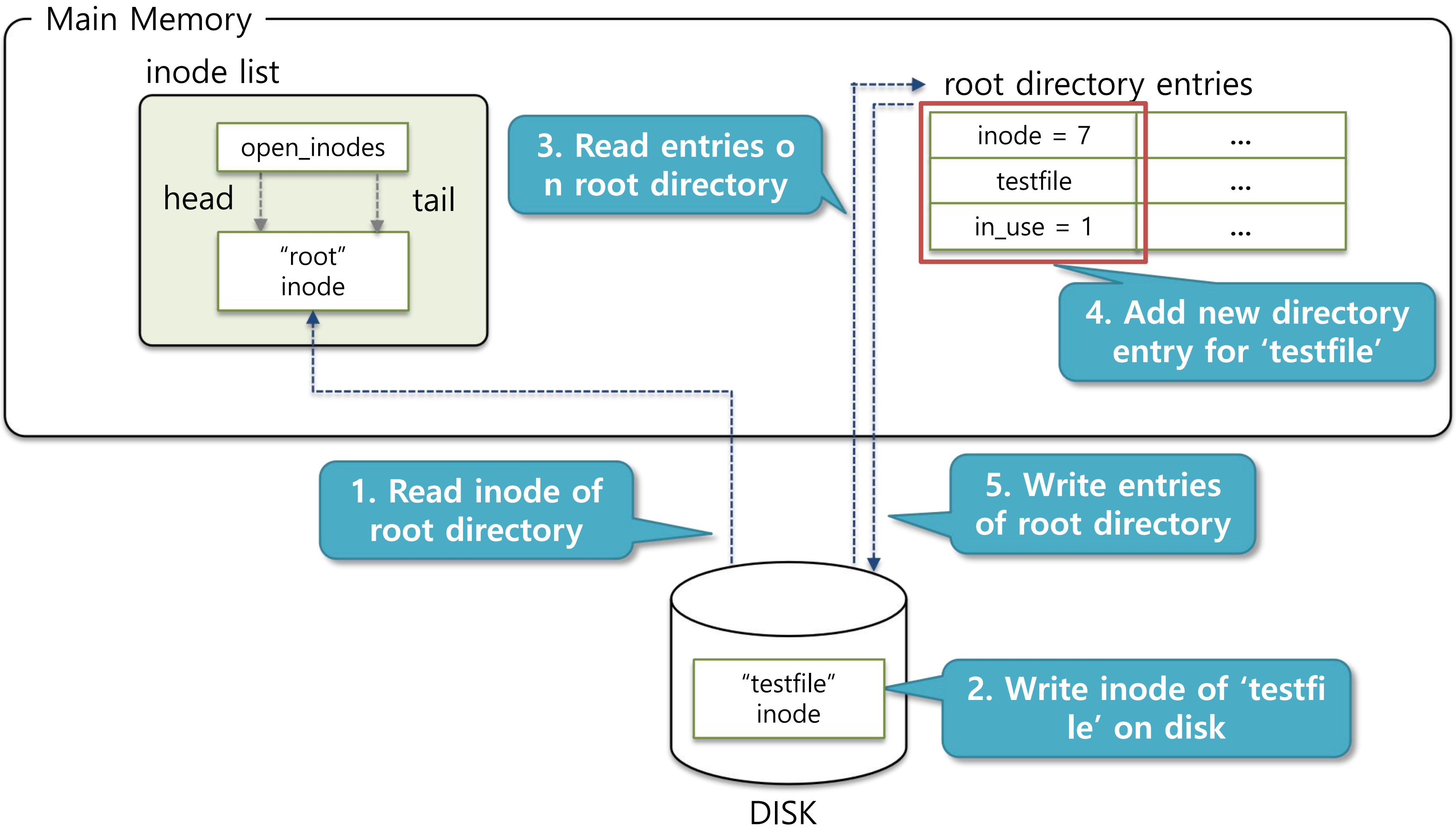
- `filesys_create()`
  - It is called by System call `'create()'`.
  - Create, initialize inode, and write it on disk.
  - Add new entry in root directory.


filesys_create()
Allocate root directo ry on memroy
Allocate new block f or inode
Write inode on disk
Add directory entry
success?
YES
NO
Deallocate block for i node
Deallocate in-memory ino de of root directory
return success;

File create

ex) 'testfile' create

```
bool filesys_create ("testfile")
```



1. Read inode of root directory

2. Write inode of 'testfi le' on disk

3. Read entries o n root directory

4. Add new directory entry for 'testfile'

5. Write entries of root directory

File create in Pintos

- `filesys_create`
  - Create and initialize inode, and add new directory entry to root directory.

pintos/src/filesys/filesys.c

```
bool filesys_create (const char *name, off_t initial_size) {
    block_sector_t inode_sector = 0;
    struct dir *dir = dir_open_root();
    bool success = (dir != NULL
                    && free_map_allocate (1, &inode_sector))
    …
```

  - `dir_open_root()` : Allocate dir structure for root directory on memory.
  - `free_map_allocate()` : Allocate 1 sector from the free map and save the start sector at inode_sector.

File create in Pintos

pintos/src/filesys/filesys.c – `filesys_create()` (Cont.)

```
               && inode_create (inode_sector, initial_size)
               && dir_add (dir, name, inode_sector));
    …
    return success;
}
```

- `inode_create()` :
  - initialize an inode with initial_size byte
  - write it on disk.
- `dir_add()`: Add new directory entry to directory.

Opening a file

- inode_open: read on-disk inode at sector and returns its pointer.

pintos/src/filesys/inode.c

```
struct inode * inode_open (block_sector_t sector) {
    …
    /* Firstly, find in-memory inode in open_inodes */
    for (e = list_begin (&open_inodes); e != list_end (&open
_inodes); e = list_next (e)) {
        inode = list_entry (e, struct inode, elem);
        if (inode->sector == sector) {
            inode_reopen (inode);
            return inode;
        }
    }
    …
```

Opening a file

pintos/src/filesys/inode.c – `inode_open()` (Cont.)

```
    …
    /* Allocate in-memory inode */
    inode = malloc (sizeof *inode);
    …
    /* Initialize in-memory inode */
    list_push_front (&open_inodes, &inode->elem);
    inode->sector = sector;
    inode->open_cnt = 1;
    inode->deny_write_cnt = 0;
    inode->removed = false;
    block_read (fs_device, inode->sector, &inode->data);
    return inode;
}
```

Opening a directory

- Allocate `dir` structure and set its field.: inode and pos (offset).

pintos/src/filesys/directory.c

```
struct dir * dir_open (struct inode *inode) {
    struct dir *dir = calloc (1, sizeof *dir);
    if (inode != NULL && dir != NULL) {
        dir->inode = inode;
        dir->pos = 0;
        return dir;
    }
    …
}
```

Allocating the free block

pintos/src/filesys/free-map.c

```
bool free_map_allocate (size_t cnt, block_sector_t *sectorp){
    block_sector_t sector = bitmap_scan_and_flip (free_map, 0, cn
t, false);
    …
    if (sector != BITMAP_ERROR)
        *sectorp = sector;
    return sector != BITMAP_ERROR;
}
```

- Find `cnt` consecutive free blocks, scanning `free-map.`
- `cnt :` the number of block to allocate
- `sectorp :` start address of blocks allocated
- `bitmap_scan_and_flip():` Find contiguous false bitmap entries and set them true.

Creating a directory

- Add `name` file to `dir`.
- Inode of the file is at sector `inode_sector`.

pintos/src/filesys/directory.c

```
bool dir_add (struct dir *dir, const char *name, block_secto
r_t inode_sector) {
    struct dir_entry e;
    off_t ofs;
    bool success = false;
    …
    /* Check that NAME is not in use. */
    if (lookup (dir, name, NULL, NULL))
        goto done;

    /* Find unused directory entry in direcctory */
    for (ofs = 0; inode_read_at (dir->inode, &e, sizeof e, o
fs) == sizeof e; ofs += sizeof e)
        if (!e.in_use)
            break;
    …
```

Background: File create in pintos (Cont.)

pintos/src/filesys/directory.c – `dir_add()` (Cont.)

```
     …
     /* Write slot. */
     e.in_use = true;
     strlcpy (e.name, name, sizeof e.name);
     e.inode_sector = inode_sector;
     success = inode_write_at (dir->inode, &e, sizeof e, ofs)
  == sizeof e;
     done:
         return success;
  }
```

Directory lookup

- lookup
  - Check if file name exist in directory or not.
  - Return address of `dir_entry` structure by parameter.

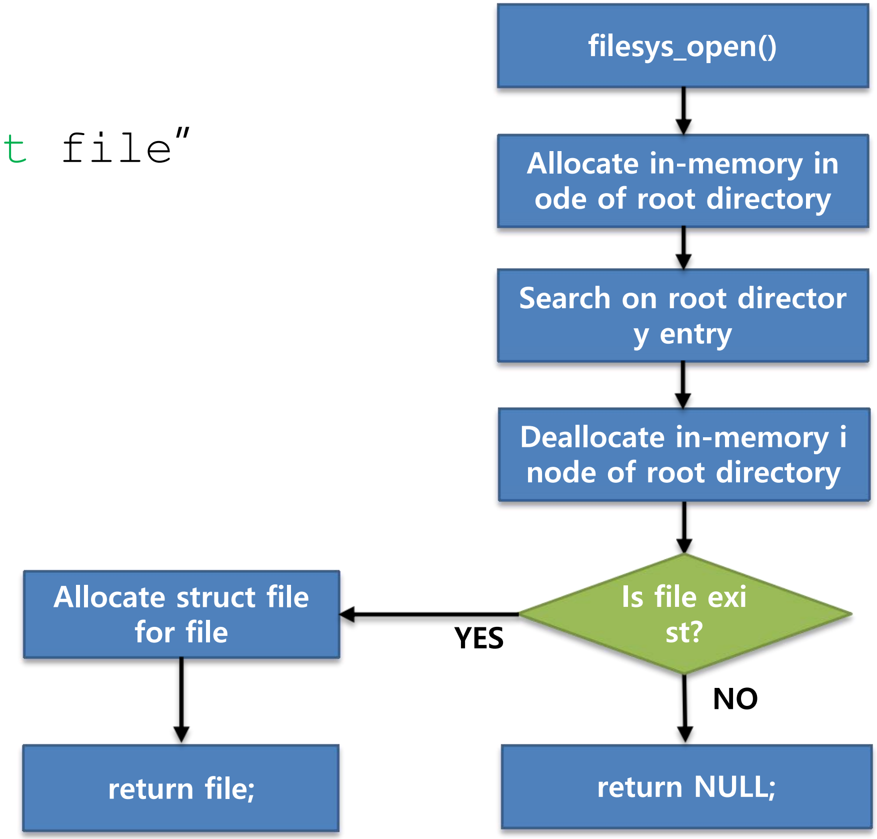
pintos/src/filesys/directory.c

```
static bool lookup (const struct dir *dir, const char *name,
        struct dir_entry *ep, off_t *ofsp) {

    struct dir_entry e;
    size_t ofs;
    …
    for (ofs = 0; inode_read_at (dir->inode, &e, sizeof e, ofs) == s
izeof e; ofs += sizeof e)
        if (e.in_use && !strcmp (name, e.name)) {
            if (ep != NULL)
                *ep = e;
            if (ofsp != NULL)
                *ofsp = ofs;
            return true;
        }
    return false;
}
```

Open a file

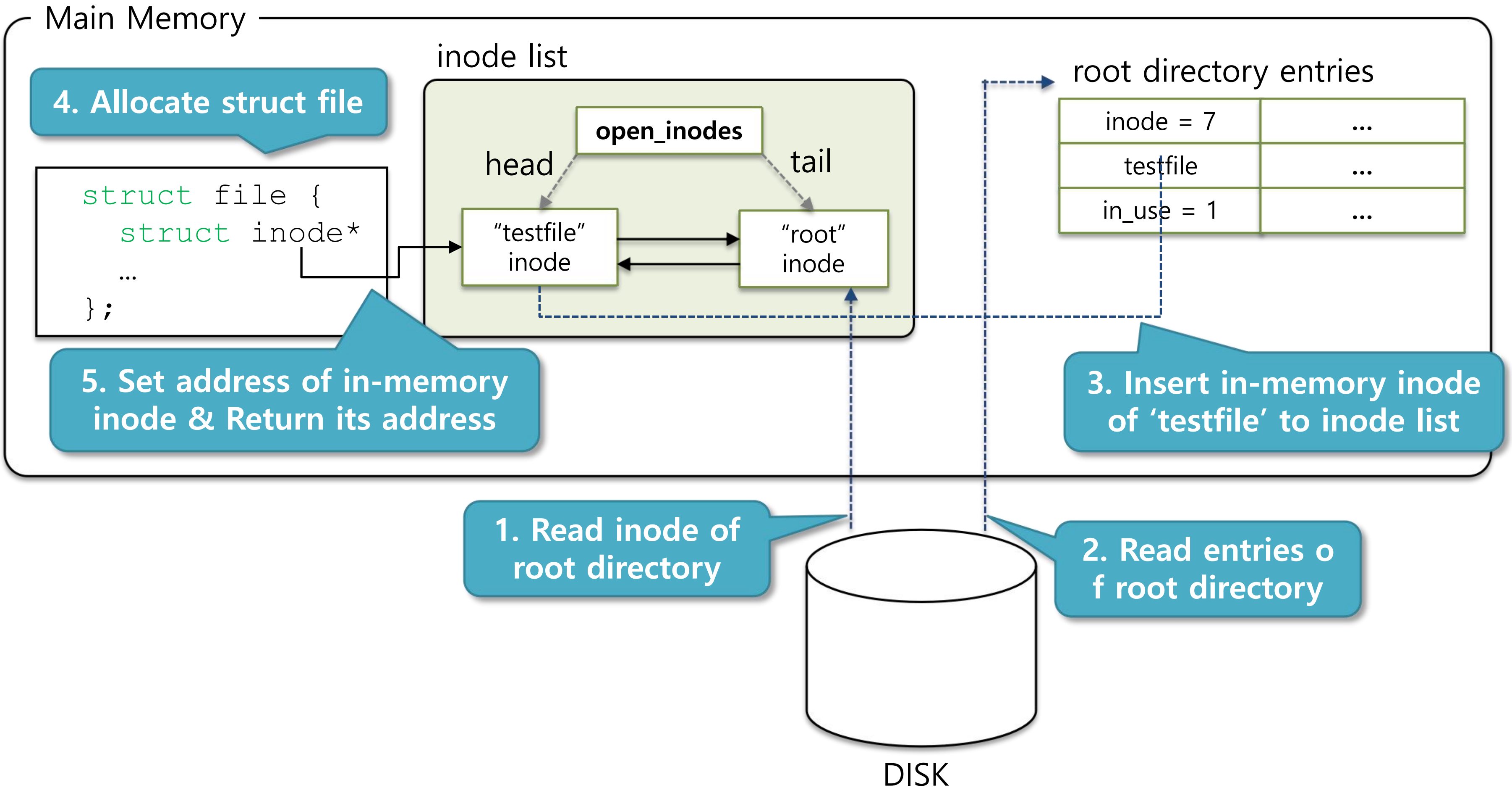
- `struct file *filesys_open(const char *name)`
  - It is called by System call ‘`open()`’.
  - Add inode to inode list.
  - Allocate and Initialize “`struct file`” and return its address.


filesys_open()
Allocate in-memory in ode of root directory
Search on root director y entry
Deallocate in-memory i node of root directory
Is file exi st?
YES
Allocate struct file for file
return file;
NO
return NULL;

Open a file

ex) ‘testfile’ open

```
struct file * filesys_open (“testfile”)
```



```
struct file {
  struct inode*
  …
};
```

inode = 7	…
testfile	…
in_use = 1	…

1. Read inode of root directory

2. Read entries o f root directory

3. Insert in-memory inode of ‘testfile’ to inode list

4. Allocate struct file

5. Set address of in-memory inode & Return its address

Open a file

- filesys_open
  - Allocate and initialize struct file, and return its address.

pintos/src/filesys/filesys.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);
}
```

  - dir_open_root() : Add in-memory inode of root directory to open_inodes list.
  - dir_lookup() : Find directory entry that have name in directory, and open it. (Allocate in-memory inode and add it to open_inodes)
  - file_open() : Allocate and initialize struct file on memory.

Open a file

- dir_lookup
  - Find file in directory, open it, and return success or not.

pintos/src/filesys/directory.c

```
bool dir_lookup (const struct dir *dir, const char *name,
                 struct inode **inode) {
     struct dir_entry e;
     …
     if (lookup (dir, name, &e, NULL))
          *inode = inode_open (e.inode_sector);
     else
          *inode = NULL;
     return *inode != NULL;
}
```

  - lookup() : Read directory from disk, find file, save its directory entry at 3r d parameter.

Open a file

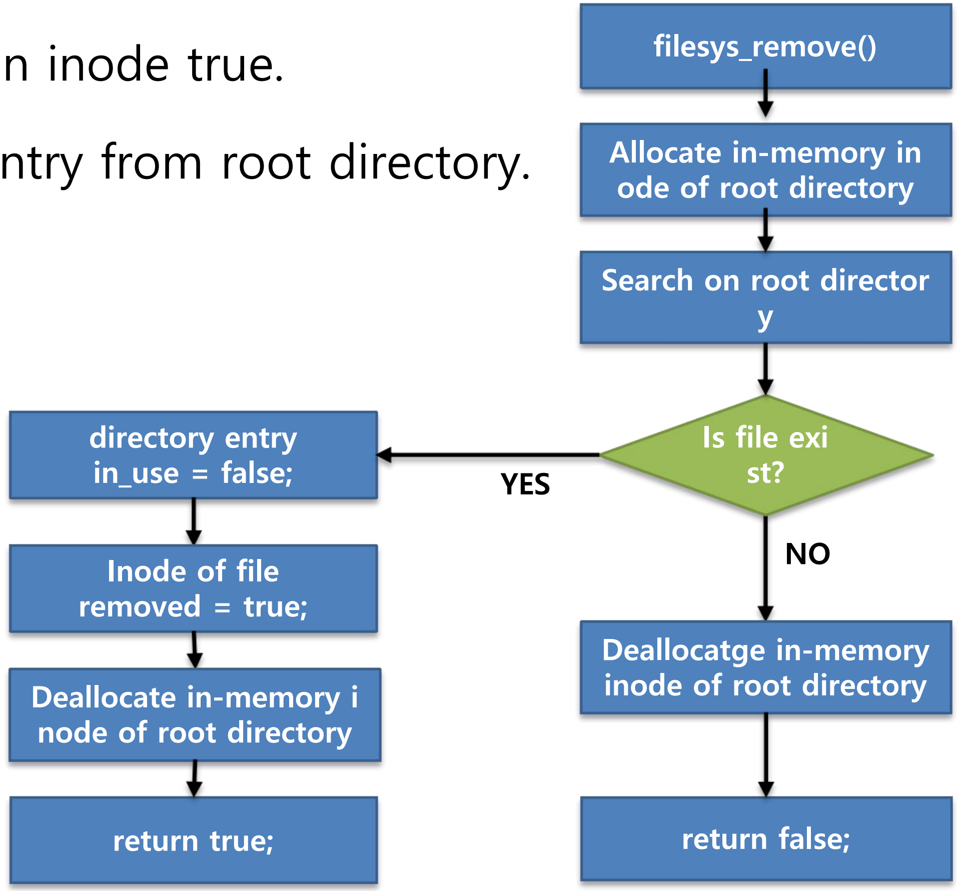
- file_open
  - Allocate and initialize struct file on memory, and return its address.

pintos/src/filesys/file.c

```
struct file * file_open (struct inode *inode) {
    /* Allocate struct file */
    struct file *file = calloc (1, sizeof *file);
    /* Initialize struct file */
    if (inode != NULL && file != NULL) {
        file->inode = inode;
        file->pos = 0;
        file->deny_write = false;
        return file;
    …
}
```

Remove a file

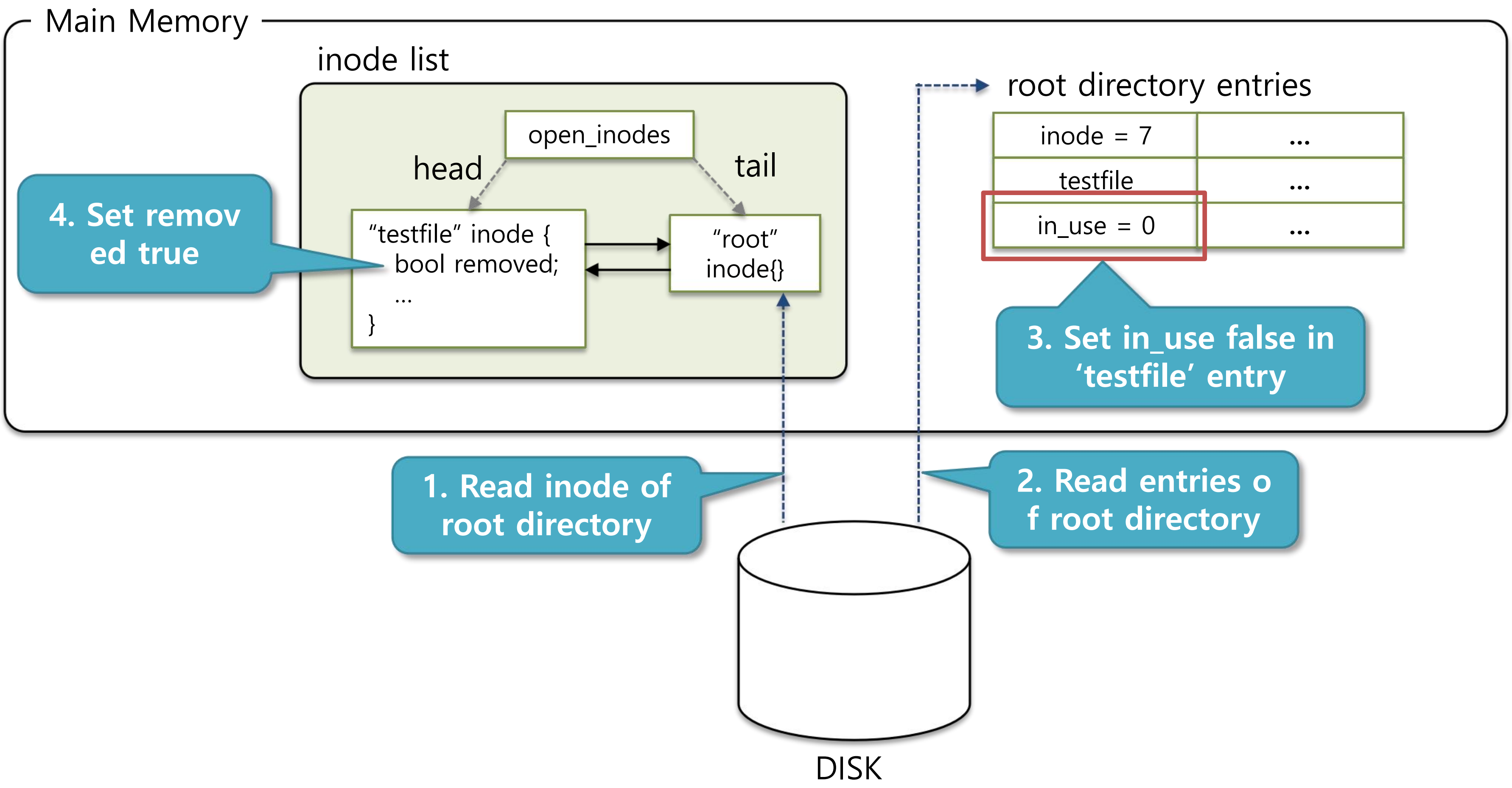
- `filesys_remove()`
  - It is called by System call ‘`remove()`’.
  - Set flag `removed` on inode true.
  - Remove directory entry from root directory.


filesys_remove()
Allocate in-memory in ode of root directory
Search on root director y
Is file exi st?
YES
directory entry in_use = false;
Inode of file removed = true;
Deallocate in-memory i node of root directory
return true;
NO
Deallocatge in-memory inode of root directory
return false;

Remove a file

ex) 'testfile' remove

```
bool filesys_remove ("testfile")
```



4. Set remov ed true

3. Set in_use false in 'testfile' entry

1. Read inode of root directory

2. Read entries o f root directory

Remove a file

- filesys_remove
  - Remove target file entry from directory.
  - Set removed flag in in-memory inode true.

pintos/src/filesys/filesys.c

```
bool filesys_remove (const char *name) {
    struct dir *dir = dir_open_root ();
    bool success = dir != NULL && dir_remove (dir, name);
    dir_close (dir);
    return success;
}
```

Remove a file

- `dir_remove`
  - Remove entry of target file in directory.
  - Set removed flag in In-memory inode true.
  - Write updated directory entry on disk.

pintos/src/filesys/directory.c

```
bool dir_remove (struct dir *dir, const char *name) {
        …
        /* Find directory entry. */
        if (!lookup (dir, name, &e, &ofs))
                goto done;
        …
        /* Erase directory entry. */
        e.in_use = false;
        if (inode_write_at (dir->inode, &e, sizeof e, ofs) != si
zeof e)
                goto done;
        …
```

Remove a file

pintos/src/filesys/directory.c – `dir_remove()`

```
    …
    /* Open inode. */
    inode = inode_open (e.inode_sector);
    …
    /* Remove inode. */
    inode_remove (inode);
    success = true;
   done:
       inode_close (inode);
       return success;
}
```

- `inode_open()` : Add in-memory inode to `open_inodes` list.
- `inode_remove()` : Set removed flag in in-memory inode true.

Summary

- Filesystem format
- Create a file
- Create a directory
- Open a file
- Remove a file.

Appendix

Type of block devices

- In Pintos, there are four types of block devices.
  - BLOCK_KERNEL: Storage to save kernel binary file.
  - BLOCK_FILESYS: Storage to be used to '/' filesystem partition.
  - BLOCK_SCRATCH: SCRATCH filesystem where files are saved to be forwarded to/from virtual machine
  - BLOCK_SWAP: Storage to be used to swap partition

Initializing the block devices (1)

threads/init.c:76

```
int
main (void)
{
  char **argv;

  /* Clear BSS. */
  bss_init ();
  ...
#ifdef FILESYS
  /* Initialize file system. */
  ide_init ();
  locate_block_devices ();
  filesys_init (format_filesys);
#endif

  printf ("Boot complete.\n");

  /* Run actions specified on kernel command line. */
  run_actions (argv);

  /* Finish up. */
  shutdown ();
  thread_exit ();
}
```

devices/ide.c:101

```
void ide_init (void)
{
  size_t chan_no;

  for (chan_no = 0; chan_no < CHANNEL_CNT; chan_no++)
    {
      struct channel *c = &channels[chan_no];
      int dev_no;

      ...

      /* Read hard disk identity information. */
      for (dev_no = 0; dev_no < 2; dev_no++)
        if (c->devices[dev_no].is_ata)
          identify_ata_device (&c->devices[dev_no]);
    }
}
```

- Find ATA device from each channel.
- Identify and register the device (`identify_ata_device()`) (next slide)

Initializing the block devices (2)

devices/ide.c:101

```
static void
identify_ata_device (struct ata_disk *d)
{
  struct channel *c = d->channel;
  char id[BLOCK_SECTOR_SIZE];
  block_sector_t capacity;
  char *model, *serial;
  char extra_info[128];
  struct block *block;

  ASSERT (d->is_ata);

  /* Send the IDENTIFY DEVICE command, wait for an interrupt
     indicating the device's response is ready, and read the data
     into our buffer. */
  select_device_wait (d);
  issue_pio_command (c, CMD_IDENTIFY_DEVICE);
  sema_down (&c->completion_wait);
  if (!wait_while_busy (d))
    {
      d->is_ata = false;
      return;
    }
  input_sector (c, id);
  ...
```

- Identify block device, using IDENTIFY_DEVICE command.
- send command to the block device and wait for completion.
- The command returns the information about block device which is saved in `id`.

Initializing the block devices (3)

```
  ...
  /* Calculate capacity.
     Read model name and serial number. */
  capacity = *(uint32_t *) &id[60 * 2];
  model = descramble_ata_string (&id[10 * 2], 20);
  serial = descramble_ata_string (&id[27 * 2], 40);
  snprintf (extra_info, sizeof extra_info,
            "model \"%s\", serial \"%s\"", model, serial);

  /* Disable access to IDE disks over 1 GB, which are likely
     physical IDE disks rather than virtual ones.  If we don't
     allow access to those, we're less likely to scribble on
     someone's important data.  You can disable this check by
     hand if you really want to do so. */
  if (capacity >= 1024 * 1024 * 1024 / BLOCK_SECTOR_SIZE)
    {
      printf ("%s: ignoring ", d->name);
      print_human_readable_size (capacity * 512);
      printf ("disk for safety\n");
      d->is_ata = false;
      return;
    }

  /* Register. */
  block = block_register (d->name, BLOCK_RAW, extra_info, capacity,
                          &ide_operations, d);
  partition_scan (block);
}
```

- Print the information about the block device.
- Prevent attaching block device bigger than 1GB.
- Register the block device.

Initializing the block devices (4)

devices/ide.c:101

```
struct block *
block_register (const char *name, enum block_type type,
                const char *extra_info, block_sector_t size,
                const struct block_operations *ops, void *aux)
{
  struct block *block = malloc (sizeof *block);
  if (block == NULL)
    PANIC ("Failed to allocate memory for block device descriptor");

  list_push_back (&all_blocks, &block->list_elem);
  strlcpy (block->name, name, sizeof block->name);
  block->type = type;
  block->size = size;
  block->ops = ops;
  block->aux = aux;
  block->read_cnt = 0;
  block->write_cnt = 0;

  printf ("%s: %'"PRDSNu" sectors (", block->name, block->size);
  print_human_readable_size ((uint64_t) block->size * BLOCK_SECTOR_SIZE);
  printf (")");
  if (extra_info != NULL)
    printf (", %s", extra_info);
  printf ("\n");

  return block;
}
```

- Create and fill the `struct block` object out.
- Push the object into `all_blocks` list.

Initializing the block devices (5)

threads/init.c:76

```
int
main (void)
{
  char **argv;

  /* Clear BSS. */
  bss_init ();
  ...
#ifdef FILESYS
  /* Initialize file system. */
  ide_init ();
  locate_block_devices ();
  filesys_init (format_filesys);
#endif

  printf ("Boot complete.\n");

  /* Run actions specified on kernel command line. */
  run_actions (argv);

  /* Finish up. */
  shutdown ();
  thread_exit ();
}
```

thread/init.c:391

```
static void locate_block_devices (void)
{
  locate_block_device (BLOCK_FILESYS, filesys_bdev_name);
  locate_block_device (BLOCK_SCRATCH, scratch_bdev_name);
  locate_block_device (BLOCK_SWAP, swap_bdev_name);
}
```

- Register three block devices; BLOCK_FILESYS, BLOCK_SCRATCH, BLOCK_SWAP.
- BLOCK_KERNEL don’t need to be registered, because kernel is already loaded.

Initializing the block devices (6)

thread/init.c:405

```
static void locate_block_device (enum block_type role, const char *name)
{
  struct block *block = NULL;

  if (name != NULL) {
      block = block_get_by_name (name);
      if (block == NULL)
        PANIC ("No such block device \"%s\"", name);
  } else {
      for (block = block_first (); block != NULL; block = block_next (block))
        if (block_type (block) == role)
          break;
  }

  if (block != NULL) {
      printf ("%s: using %s\n", block_type_name (role), block_name (block));
      block_set_role (role, block);
  }
}
```

devices/block.c:61

```
void
block_set_role (enum block_type role, struct block *block)
{
  ASSERT (role < BLOCK_ROLE_CNT);
  block_by_role[role] = block;
}
```

- Find `struct block` object by name or type.
- Set `block_by_role[]` array with the address of the `struct block` object.