Pintos lab4

Youngjin Kwon

Contents

- Implementing file indexing APIs (FAT)
- Building hierarchical namespace
  - Hierarchical directory
- Modifying block access to use FAT
- Softlink

Indexing structure: FAT

- Formatting file system

```
void
filesys_init (bool format) {
  filesys_disk = disk_get (0, 1);
  if (filesys_disk == NULL)
    PANIC ("hd0:1 (hdb) not present

  inode_init ();

#ifdef EFILESYS
  fat_init ();

  if (format)
    do_format ();
```

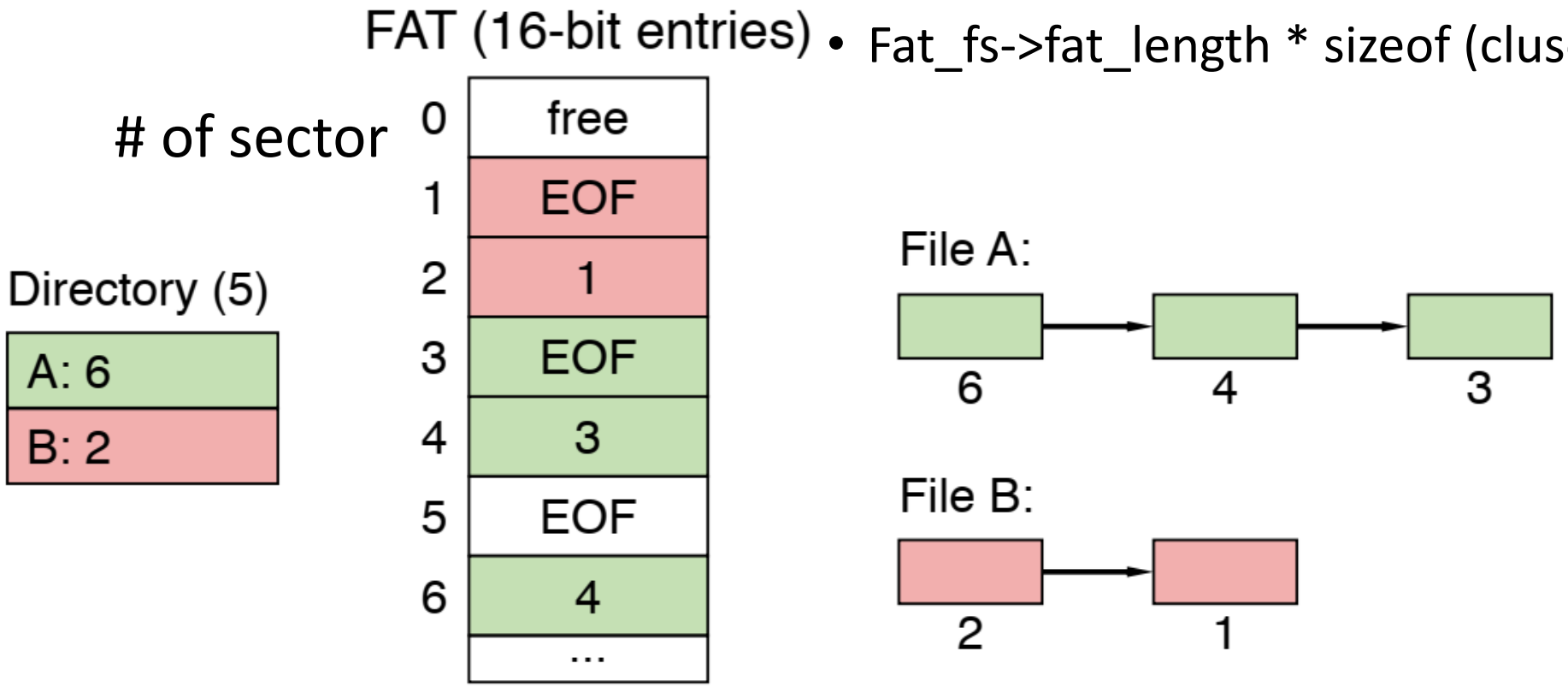
```
/* Formats the file system. */
static void
do_format (void) {
  printf ("Formatting file system...");

#ifdef EFILESYS
  /* Create FAT and save it to the disk. */
  fat_create ();
  fat_close ();
#else
  free_map_create ();
  if (!dir_create (ROOT_DIR_SECTOR, 16))
    PANIC ("root directory creation failed");
  free_map_close ();
#endif

  printf ("done.\n");
}
```

```
void
fat_create (void) {
  // Create FAT boot
  fat_boot_create ();
  fat_fs_init ();

  // Create FAT table
  fat_fs->fat = calloc (fat_fs->fat_length, sizeof (cluster_t));
  if (fat_fs->fat == NULL)
    PANIC ("FAT creation failed");

  // Set up ROOT_DIR_CLST
  fat_put (ROOT_DIR_CLUSTER, EOChain);

  // Fill up ROOT_DIR_CLUSTER region with 0
  uint8_t *buf = calloc (1, DISK_SECTOR_SIZE);
  if (buf == NULL)
    PANIC ("FAT create failed due to OOM");
  disk_write (filesys_disk, cluster_to_sector (ROOT_DIR_CLUSTER), buf);
  free (buf);
}

void
fat_boot_create (void) {
  unsigned int fat_sectors =
      (disk_size (filesys_disk) - 1)
      / (DISK_SECTOR_SIZE / sizeof (cluster_t) * SECTORS_PER_CLUSTER + 1) + 1;
  fat_fs->bs = (struct fat_boot){
      .magic = FAT_MAGIC,
      .sectors_per_cluster = SECTORS_PER_CLUSTER,
      .total_sectors = disk_size (filesys_disk),
      .fat_start = 1,
      .fat_sectors = fat_sectors,
      .root_dir_cluster = ROOT_DIR_CLUSTER,
```

```
#define data_start(FATBS) (1 + ((FATBS).fat_sectors))
#define data_sectors(FATBS) (((FATBS).total_sectors) - (data_start (FATBS)))
```

FAT review

- Size of fat table?
- Fat_fs->fat_length * sizeof (cluser_t)



Implement APIs in filesys/fat.c described in the project document

Hierarchical namespace (directory hierarchy)

- Current directory
  - Where to store the current directory? Struct thread. Use struct dir..
- You need to implement changing the current directory (e.g., the cd command)
  - Expand dir_create() to add . (current directory) and .. (parent directory) to each directory
  - Closely look at dir_add() or dir_remove() to figure out how directories are organized

Directory

- A directory is array of struct dir_entry

```
/* A single directory entry. */
struct dir_entry {
  disk_sector_t inode_sector;         /* Sector number of header. */
  char name[NAME_MAX + 1];            /* Null terminated file name. */
  bool in_use;                        /* In use or free? */
};
```

```
bool
filesys_create (const char *name, off_t initial_size) {
  disk_sector_t inode_sector = 0;
  struct dir *dir = dir_open_root ();
  bool success = (dir != NULL
      && free_map_allocate (1, &inode_sector)
      && inode_create (inode_sector, initial_size)
      && dir_add (dir, name, inode_sector));
  if (!success && inode_sector != 0)
    free_map_release (inode_sector, 1);
  dir_close (dir);

  return success;
}
```

```
bool
dir_add (struct dir *dir, const char *name, disk_sector_t inode_sector) {
  struct dir_entry e;
  off_t ofs;
  bool success = false;

  ASSERT (dir != NULL);
  ASSERT (name != NULL);

  /* Check NAME for validity. */
  if (*name == '\0' || strlen (name) > NAME_MAX)
    return false;

  /* Check that NAME is not in use. */
  if (lookup (dir, name, NULL, NULL))
    goto done;

  /* Set OFS to offset of free slot.
   * If there are no free slots, then it will be set to the
   * current end-of-file.

   * inode_read_at() will only return a short read at end of file.
   * Otherwise, we'd need to verify that we didn't get a short
   * read due to something intermittent such as low memory. */
  for (ofs = 0; inode_read_at (dir->inode, &e, sizeof e, ofs) == sizeof e;
       ofs += sizeof e)
    if (!e.in_use)
      break;

  /* 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;
}
```

read

modify

write

```
off_t
inode_write_at (struct inode *inode, const void *buffer_, off_t size,
    off_t offset) {
  const uint8_t *buffer = buffer_;
  off_t bytes_written = 0;
  uint8_t *bounce = NULL;

  if (inode->deny_write_cnt)
    return 0;

  while (size > 0) {
    /* Sector to write, starting byte offset within sector. */
    disk_sector_t sector_idx = byte_to_sector (inode, offset);
    int sector_ofs = offset % DISK_SECTOR_SIZE;

    /* Bytes left in inode, bytes left in sector, lesser of the two. */
    off_t inode_left = inode_length (inode) - offset;
    int sector_left = DISK_SECTOR_SIZE - sector_ofs;
    int min_left = inode_left < sector_left ? inode_left : sector_left;

    /* Number of bytes to actually write into this sector. */
    int chunk_size = size < min_left ? size : min_left;
    if (chunk_size <= 0)
      break;

    if (sector_ofs == 0 && chunk_size == DISK_SECTOR_SIZE) {
      /* Write full sector directly to disk. */
      disk_write (filesys_disk, sector_idx, buffer + bytes_written);
    } else {
      /* We need a bounce buffer. */
      if (bounce == NULL) {
        bounce = malloc (DISK_SECTOR_SIZE);
        if (bounce == NULL)
          break;
      }

      /* If the sector contains data before or after the chunk
         we're writing, then we need to read in the sector
         first.  Otherwise we start with a sector of all zeros. */
      if (sector_ofs > 0 || chunk_size < sector_left)
        disk_read (filesys_disk, sector_idx, bounce);
      else
        memset (bounce, 0, DISK_SECTOR_SIZE);
      memcpy (bounce + sector_ofs, buffer + bytes_written, chunk_size);
      disk_write (filesys_disk, sector_idx, bounce);
    }
```

Handle the case where directory and file grows

Write data to disk using FAT

```
static disk_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 / DISK_SECTOR_SIZE;
  else
    return -1;
}
```

Update this part to use FAT you implement

Also, modify inode_create() which creates new inode sector and inode_open() which read inode sectors to use FAT

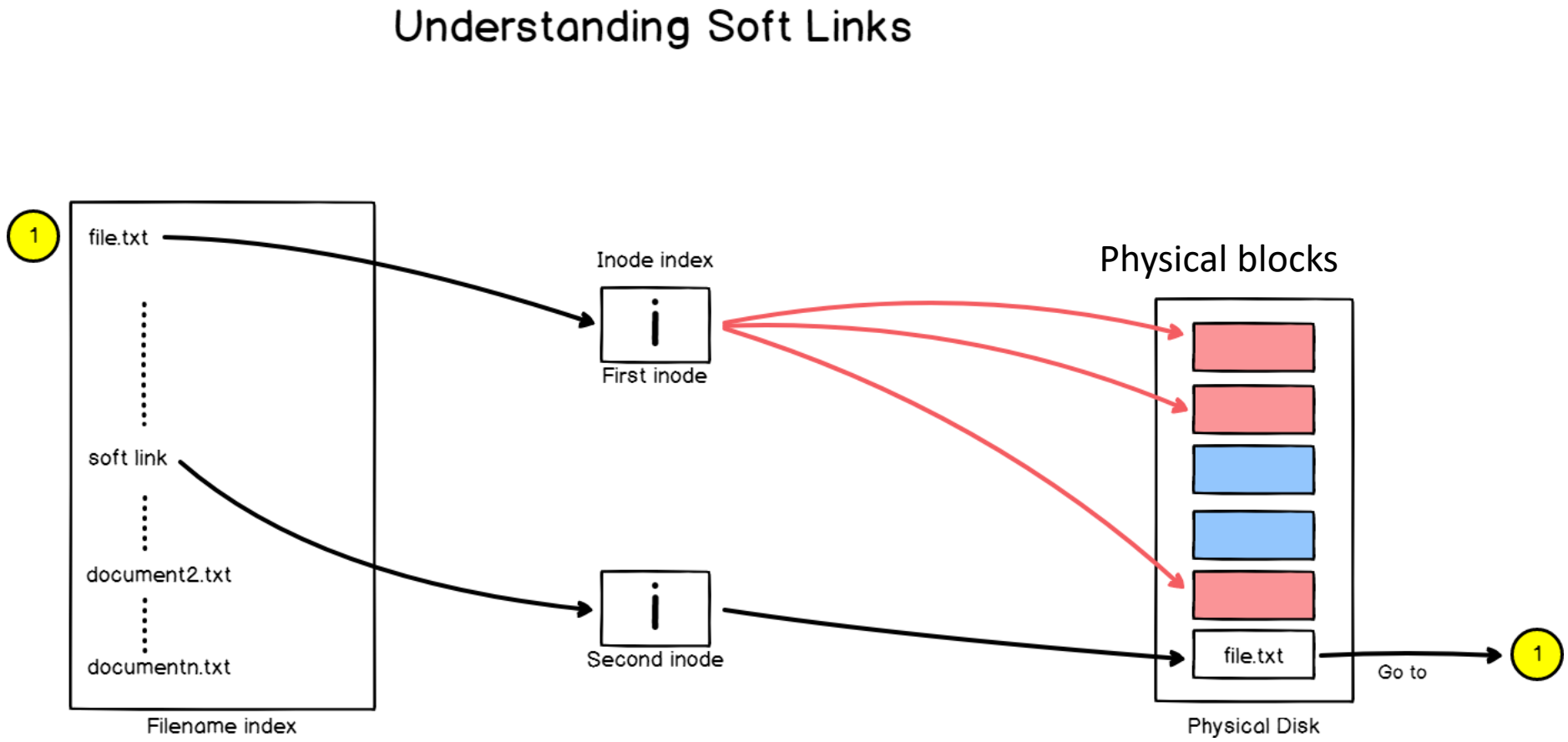
Open a file (or directory) with absolute path

- Open(“/a/b/c/d”) or remove(“/a/b/c/d”)
  - File system should walk directories from “/” to “/a/b/c/” to find d

```
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);
}
```

- Extend filesys_open() and filesys_remove() to walk a directory hierarchy given by *name

Softlink


Understanding Soft Links
1
file.txt
soft link
document2.txt
documentn.txt
Filename index
Inode index
First inode
Second inode
Physical blocks
file.txt
Physical Disk
Go to
1

Extend inode to support soft link

- Symlink creates a new inode
  - Contents are duplicated from the target inode
  - Treat symlink inode as a normal inode
    - E.g., symlink name is in a directory, and inode # points to a symlink inode
    - Deletion of symlink does not delete file
    - Symlink can point to a directory
    - Symlink remains even if a target file is deleted (dangling symlink)

```
devconnected@debian-10:~$ ls -l shortcut
lrwxrwxrwx 1 devconnected devconnected 8 Aug 13 16:18 shortcut -> file.txt
devconnected@debian-10:~$ rm file.txt
devconnected@debian-10:~$ ls -l shortcut
lrwxrwxrwx 1 devconnected devconnected 8 Aug 13 16:18 shortcut -> file.txt
```