

CS342301: Operating System

Machine Problem 4 - File System

Deadline: 2023/1/15 23:59

1. Problem Description

NachOS native file system (NachOS FS) only supports up to 4KB file size and only has a root directory. In this assignment, you need to study NachOS FS and find out the reason for the limitations. Also, you are required to enhance the file system to let NachOS support larger file sizes and subdirectory structures.

2. Assignment

Part I. Understanding NachOS file system

Trace the file system call **before modifying the code** and use **unmodified code** to answer the following questions :

- (1) Explain how the NachOS FS manages and finds free block space? Where is this information stored on the raw disk (which sector)?
- (2) What is the maximum disk size that can be handled by the current implementation? Explain why.
- (3) Explain how the NachOS FS manages the directory data structure? Where is this information stored on the raw disk (which sector)?
- (4) Explain what information is stored in an inode, and use a figure to illustrate the disk allocation scheme of the current implementation.
- (5) Why is a file limited to 4KB in the current implementation?

Part II. Modify the file system code to support file I/O system calls and larger file size

- (1) Combine your MP1 file system call interface with NachOS FS
For your implementation simplicity, you may assume that all test cases do not contain any messy operations. (E.g.create a file that already exists, read/write exceeds the file size, etc.)

- (2) Implement five system calls:

★ `int Create(char *name, int size);`

Create a file with the `name` and with `size` bytes in the root directory. The name of the file only contains characters [A-Za-z0-9.] and with a length not greater than 9. Here, this operation will always succeed and return 1.

★ `OpenFileId Open(char *name);`

Open the file with `name` and return its `OpenFileId`. **Only at most one file will be opened at the same time.** Here, any `OpenFileId` larger than 0 is considered a successful open. (You do not need to maintain `OpenFileTable` in this assignment)

★ `int Read(char *buf, int size, OpenFileId id);`

★ `int Write(char *buf, int size, OpenFileId id);`

Read/Write `size` characters from/to the file to/from `buf`. Return the number of characters actually read/written from/to the file. Here, `id` will always be valid and no messy operations will be given.

★ `int Close(OpenFileId id);`

Close the file by `id`. Here, this operation will always succeed and return 1.

(3) Enhance the FS to let it support up to 32KB file size

You can use any approach including modifying the allocation scheme or extending the data block pointer structure, etc.

Important: You ARE NOT allowed to change the sector size!!!

For your implementation simplicity, you may assume that all of the operations will not be messy. (E.g. copy a file larger than 32KB, try to print a non-existing file, etc.)

- **Verification:**

We will use these commands to check your correctness:

```
> ../build.linux/nachos -f
```

Format the disk on NachOS.

```
> ../build.linux/nachos -cp <file_to_be_copied> <destination_on_NachOS_FS>
```

Copy a file from Linux FS to NachOS FS.

```
> ../build.linux/nachos -p <file_to_be_dumped>
```

Print the content of a file on NachOS disk.

Part III. Modify the file system code to support the subdirectory

For your implementation simplicity, you may assume that all the operations will not be messy. (E.g. remove a non-existing file, copy a file into a non-existing directory, create a directory in a non-existing directory, list a file instead of a directory, etc.)

(1) Implement the subdirectory structure

- ★ Use '/' as the path name separator

- ★ Use '/' at the end of the directory path

- ★ The path has a maximum length of 255

- ★ **The length of the directory and file name does not exceed 9.**

- ★ **All paths are absolute** (e.g. /testing/num100, /1000, etc.)

(2) Support up to 64 files/subdirectories per directory

- **Verification:**

```
> ../build.linux/nachos -f
```

```
> ../build.linux/nachos -mkdir <directory_to_be_created>
```

```
> ../build.linux/nachos -cp <file_to_be_copied> <absolute_path_on_NachOS_FS>
```

```
> ../build.linux/nachos -r <file_to_be_deleted>
```

Delete a file (not a directory) from NachOS FS

```
> ../build.linux/nachos -l <list_directory>
```

List the file/directory in a **directory**.

```
> ../build.linux/nachos -p <file_to_be_dumped>
```

```
> ../build.linux/nachos -lr <directory_to_be_listed>
```

Recursively list the file/directory in a directory. The Directory always exists.

Example:

```
> nachos -lr /
```

```
[D] dir1
```

```
  [F] file1
```

```
  [D] dir2
```

```
    [F] file3
```

```
    [F] file2
```

```
  [F] file4
```

The output format needs to show:

1. The object is a file or a directory.

2. The relationship between the directory and the file. **You should use indentation as in the example above. Otherwise, you will lose some points.**

3. Bonus Assignment

There are no output format limitations, any way to show your result is acceptable. You do not have to handle messy operations. (E.g. copy a file larger than 64MB, delete/list a non-existing directory).

- Bonus I: Enhance the NachOS to support even larger file size
 - Extend the disk from 128KB to 64MB
 - Support up to 64 MB single file
- Bonus II: Multi-level header size
 - Show that smaller files can have smaller header size
 - Implement at least 3 different sizes of headers for different sizes of files
 - **Design your own test cases to show your implementation is correct.**
 - **The test cases need to print the memory space occupied by the file header.**
 - **The test cases need to explain that your implementation can support at least 3 different sizes of headers.**
- Bonus III. Recursive operations on directories
 - Support recursive removal of a directory

We will use these commands to check your correctness:

```
> ../build.linux/nachos -rr <file/directory_to_be_removed>
```

Remove the file or recursively remove the directory

The directory to be removed will always exist and will not be the root directory.

4. Grading

- Implementation correctness -- 60%
 - Part II -- 30%
 - Part III -- 30%
- Report -- 20%
- Demo -- 20%
- Bonus -- 15% (5% each)

5. Reminder

- (1) eeclass
 - Please upload your Part I and implementation report in PDF format with the filename **MP4_report_<Group Number>.pdf** to eeclass.
 - You do not have to upload your NachOS to eeclass. Instead, make sure your NachOS folder is named '**NachOS-4.0_MP4**' and is under your home directory (i.e. **~/NachOS-4.0_MP4/**)
- (2) Demo policy
 - All demos will be performed on our server.
 - You are responsible for making sure your NachOS works on our server.
- (3) Please refer to the syllabus for the late submission penalty.
- (4) If you are facing a disk space not enough problem, you can try to delete the **~/cache/** directory to make a space.
- (5) 0 will be given to cheaters. Do not copy & paste!
- (6) Feel free to ask questions on eeclass!