

CMPS455 Operating Systems Term Project

Phase 3: A Simple File System

Objective In this phase of your project you will learn how to develop a basic file system in an operating system. You are asked to enhance the Nachos file system by:

1. Modifying the existing system to support large files
2. Implementing a hierarchical directory structure
3. Implementing some basic file operation commands (e.g.: `cd`, `ls` etc.)

Notes to Students Phase 3 is a Group Project. You will develop codes in the following directories:

`threads`, `test`, `fileSYS`, `userprog`

The arrangement of this document will be divided into the following sections:

- Preparation: Required readings and the concept of *shell* in Nachos will be stated.
- Design: The specification of the major components of this project will be stated. Comments regarding the relationship of this phase with previous phases will also be found in this section.
- Implementation: A number of suggestions regarding implementation details will be stated here. Note that each group should have their *own design* and may not follow the suggestions stated in that section.

It is **important** to read through this document and all the background readings before developing your design. It is also **important** to have a definite design before doing any implementation work.

Preparation

- Required readings:
 1. Read Chapter 11, 12 and all other supplementary materials
 2. A Road Map Through Nachos
 3. Salsa: An Operating Systems Tutorial
 4. Read and study the following files (at least):
`exception.cc`, `system.cc/h`, `thread.cc/h`, `syscall.h`, `filehdr.cc/h`,
`openfile.cc/h`, `fileSYS.cc/h`, `directory.cc/h`, `start.s`
and *possibly* others
- *shell*: A *shell* is a command interpreter that acts as an interface between users and the operating system. In this project, you will be provided with a shell named `newshell.c` that works correctly. This shell reads a command from the user via the console and then runs the command by invoking the kernel system call `Exec()`. For example, if you run the command `cd`, it will first invoke the system call `Exec('cd')` by the shell. `Exec('cd')` will invoke another system call `cd()`.

Design

1. (Supporting large files in nachos) Modify the file system so that the maximum size of a file can be as large as the disk size. In the basic file system (32 x 32 sectors with sector size of 256 bytes) each file is limited to a file size of just under 16K bytes. Each file has a header class FileHeader that is a table of direct pointers to the disk blocks for that file. Since the header is stored in one disk sector, the maximum size of a file is limited by the number of pointers that will be in one disk sector. Implement doubly indirect blocks so that the maximum file size can be increased to 256K.
2. (Directory Structure)
In the basic file system, all files are listed in a single directory. Modify this to allow directories to point to either files or other directories. You need routines to parse pathnames into a sequence of directories needed to find the file. Your implementation should support both the relative and absolute pathnames.

For example, for absolute pathnames:

```
/food/fromtree/apple
/drinks
```

and for relative pathnames (assuming you are in the directory /home/su/)

```
prompt> cd .. // This will go one step up. i.e. you will be in /home/

prompt> cd ../me/ // This will take you to /home/me/
```

Each directory (see `directoryEntry` and `directory` classes) will have a series of entries. Each entry can be a file or another directory. A special flag in `directoryEntry` class will mark entries that are directories.

3. (File System Commands)

These can be implemented as *system calls* and run from the shell. All groups should attempt all work stated here.

 - (a) Unless otherwise stated, all the utility commands should work the way they work on the UCS machines. Look at the man pages regarding the definition of the above commands in a UCS terminal
 - (b) The `ls` should show file names. When used with the `-l` option it should also show size, type (file or directory) and date of modification of file. When used with `-R` option, it should show all the files, directories under the current directory *recursively*. For example, if the subdirectories contains other directories, the files in those directories should also be listed.
 - (c) The `cat` command should work in ONLY one way: `cat <filename>`. It will display the contents of the file.
 - (d) You need to store and manipulate the date of last access to the file. That is, a time field should be in the Directory Entry structure. It is modified if the file is written or last accessed.
 - (e) The `mkdir` command should have the format `mkdir path/directory`. This allows the creation of a directory.
 - (f) The `rmdir` command should have the format `rmdir path/directory`. This will remove a directory from the filesystem.

- (g) The `cd` command should have the format `cd pathname`. This will allow the user to change the current working directory. The pathname can be either a relative OR absolute pathname.
- (h) The `pwd` command takes no parameters. It will display the pathname of the current working directory.
- (i) The `rm` command should have the format `rm path/filename`. This will allow the user to remove a file from anywhere in the nachos filesystem. Implement the `-R` option also. That is, the command `rm -R` will remove all the contents of the current directory *recursively*.
- (j) The `cp` command should have the format `cp path1/filename1 path2/filename2`. This will copy a file to a new location.
- (k) Note:
 - i. Implement the commands and flags mentioned above only. **There is no need to add any flags (unmentioned) to the above commands.** For example, `ls -a` would be unnecessary.
 - ii. To avoid multiple interpretation of the requirements and specification, all questions regarding this phase should be directed to our [faq](#) section. Check the [faq](#) often for any possible updates.

Implementation

We state a number of suggestions regarding implementation issues. There could be some other ways to do it. You are welcome to use your own ideas.

1. The directory structure in nachos file system is implemented as a file. One directory has a file header and some data sectors (store the directory entry table of this directory). You can look at the `directory.h/directory.cc` file to get more information about it. But be aware that the original version of the directory class in nachos can only hold up to 5 directory entries. Because the directory entry table could be dynamic, you need to deal with the growing and shrinking operations to the directory.
2. Each directory contains the “.” and “..” directory entries which represent itself and its parent directory, except for the root directory whose parent directory is itself.
3. Root directory is a special directory. It does not have parent directory.

Testing

All system commands including available options should be tested. Your implementation should be robust enough to handle cases with many different combinations. For example, your `cd` command should handle the cases like `cd /drinks` and `cd /drinks/../../drinks` (notice that these two commands will get into the same directory).

Submissions Guidelines/policies The following are the requirements on submissions for the tasks in phase 3. First, your final compiled executable should be ready in your

```
/w1/cs4551/xxxxnnn/nachos/code/filesys
```

you may also have modified source code in the following directories

```
/w1/cs4551/xxxxnnn/nachos/code/userprog
/w1/cs4551/xxxxnnn/nachos/code/test
/w1/cs4551/xxxxnnn/nachos/code/thread
```

Note the following:

1. The code freeze time is 5:00 PM on Wednesday, November 30, 2005. We will execute them to see that they work properly. For your printouts, submit to our office (CC 411) no later than 4:00 pm on Wednesday, November 30, 2005.
2. We need a copy of any files you have modified. Please staple them together. Don't forget to include your name and login ID number on your assignment.
3. You must also submit a report with your printouts. The report should contain descriptions of how you implemented each part of the assignment along with details on some of the data structures. The report should be about 4-5 pages long, but no less than 3 pages. It must not exceed 10 pages.
4. There will be NO EXTENSION for due dates. Work submitted after the above time will NOT be graded.