Programming Assignment 3: Unix-like File System

Update: The FAQ section is added at the end and will be updated for this project periodically.

Overview:

In this project you will implement a simple UNIX-like file system with a hierarchical directory structure. Files and directories can grow and shrink, and you need to manage free disk space. Your file system will be able to allow users to browse the directory structure, create and delete new files and directories, etc. The functionality is similar to UNIX file systems, but permission control and user management are not included. Neither do we require concurrency or high performance of your file system.

Details:

The file system emulates a Unix file system environment through an interactive shell for the user. Specifically, you are required to implement several Unix commands in the shell to allow interactive use. The shell should also support reading commands from a file, or writing output to a file, among other tasks.

File system implementation

The file system assumes that a virtual hard disk is available for you to use. Specifically, this virtual hard disk is going to be large file with a fixed size. For this project, you need to create a file of size 100MB to simulate this virtual hard disk. You can read, write, or seek this file as you will do to a hard drive. It is, however, your responsibility to organize this virtual hard disk contents so that it will be formatted into a file system and to store user files.

Shell Commands

Below is a list of shell commands that you need to support in your file system. We will test your system using the following commands. You may add some more for your own purpose. But don't change the format of the given commands. Note that other than commands cd and link, all other commands are operating on files in the current directory.

Shell commands	Arguments	Description
mkfs		Make a new file system, i.e., format the disk so that it is ready for other file system operations.
open	<filename> <flag>	Open a file with the given <flag>, return a file descriptor <fd> associated with this file. <flag>: 1: “r”; 2: “w”, 3: “rw” The current file offset will be 0 when the file is opened. If a file does not exist, and it is opened for “w”, then it will be created with a size of 0. This command should print an integer as the fd of the file. Example: open foo w shell returns SUCCESS, fd=5
read	<fd> <size>	Read <size> bytes from the file associated with <fd>, from current file offset. The current file offset will move forward <size> bytes after read. Example: read 5 10 shell returns the contents of the file (assuming it has been written)
write	<fd> <string>	Write <string> into file associated with <fd>, from current file offset. The current file offset will move forward the size of the string after write. Here <string> must be formatted as a string. If the end of the file is reached, the size of the file will be increased. Example: write 5 “hello, world”
seek	<fd> <offset>	Move the current file offset associated with <fd> to a new file offset at <offset>. The <offset> means the number of bytes from the beginning of the file. Example: seek 5 10
close	<fd>	Close the file associated with <fd>. Example: close 5
mkdir	<dirname>	Create a sub-directory <dirname> under the current directory. Example: mkdir foo

rmdir	<dirname>	Remove the sub-directory <dirname>. This directory has to be empty when it is removed. Example: rmdir foo
cd	<dirname>	Change the current directory to <dirname>. Example: cd ../../foo/bar
link	<src> <dest>	Create a link named <dest> to an existing file or directory named <src>. Note that <dest> and <src> may not be in the same directory. Both <src> and <dest> are files, not directories. Example: link foo/src ../bar/dest
unlink	<name>	Remove a link to the name. (When the link count drops to 0, delete the file). Example: unlink dest
stat	<name>	Show the status of the file or directory with name <name>. It should display its inode information; whether it is a file or directory; its link count; size of the file/directory; number of blocks allocated; other information stored about this file/directory. Example: stat foo
ls		Show the content of the current directory. No parameters need to be supported.
cat	<filename>	Show the content of the file. Example: cat foo
cp	<src> <dest>	Copy the file from a source location to the destination location. Note that <src> and <dest> may not be in the same directory. If the <dest> already exists, the program should report an error. Example: cp foo /../bar/dest
tree		List the contents of the current directory in a tree-format. For each file listed, its date file size should be included. To understand this command better, you may refer to this command output under the command line shell in a Windows system.

import	<srcname> <destname>	Import a file from the host machine file system to the current directory. Example: import /d/foo.txt foo.txt This command copies a file located in the D drive of the host machine to the current directory.
export	<srcname> <destname>	Export a file from the current directory to the host machine file system. Example: export foo.txt /d/foo.txt This command copies the file in the current directory to the D drive of the host machine.

Extra Credit (15 points)

Implement remote file shell and its operations. Optionally you may use the RPC program you developed for PA1 for this goal, but this is not required. A remote shell allows the user to start a file system server on one machine, and use the client to remotely connect and operate on the file system. User log-in and log-out do not need to be supported.

Submissions

1) Design document must be submitted with your code together. The document should specify how to use the implemented system, including the names of implemented commands and functions. (20pt)
2) The source program. The program can be written in C or C++. It is not recommended for you to use Java, as it does not provide good functionality for handling pointers. But if you want you can still write the program in Java. Your project should be able to run under the Linux environment or the Windows environment. You need to specify which development environment you have chosen in your design document. (80pt)

Testing

For our testing purposes, your program should support redirection. Specifically, it should be able to read commands from a file, and write output into a file. For example, following line of command (invoked in your shell) will generate the testing output, assuming that the shell is invoked via sh. However, you don't need to start a new process though. Your shell can directly parse the input and generate output as needed.

sh < testinput > testoutput

For example, we will use a test script for grading like following:

sh < gradingscript > gradingoutput

You can also develop something similar. However, you should document how you actually use this redirection in your design document for submission. We are going to use a standard testing script for testing your program, but it is your responsibility to test your program thoroughly before submission.

FAQ:

Q: How to implement the file system internal structure?

A: You are expected to implement it using a similar architecture as the inode approach in Unix file system. Some details on the inode file system data structures can be found at: https://en.wikipedia.org/wiki/Inode_pointer_structure

Q: How is the disk created?

A: You can star the shell and type in the “mkfs” command for the first time to format a new disk. Note that if you call this command again it will reformat the disk.

Q: Do I need to start the shell to type commands?

A: Yes. The shell should be started first before commands are typed. Note that the shell should also support batch processing, but such support can be either implemented in the shell or outside the shell. That is, suppose that you shell is called “sh”, then you can either start the shell and then start the batch processing (after importing the file into the disk), or simply type “sh < commands” in the command line to process the commands.

Q: What types of files can “import” and “export” support?

A: We will only test text files.

Q: How to get the extra credit?

A: It is up to you. As long as your program can invoke a remote server for the operations on the remote disk file that will be fine. It is up to you to decide if you have the functionality implemented on the client or the server side.