CS355 Final Project README

Yue Chen, Paprika Chen

format.c

usage:

```
  usage: ./format <filename> [-s size_mb]
  (you can run with only the filename to create a disk image with a
default size of 1mb, or run with flag -s to specify the size you want.
[caution: Since we will read all content from disk into RAM, the size
of the disk can't be larger than 1gig)
```

The disk structure:

after running `./format DISK`:

```
root/
-file.txt
-layerone/
  -layertwo/
   -path.txt
   -more.txt
```

(include a path, can be used to test redirection input)

(include a large txt, can be used to test cat and more)

User inode/user table:

Right after the superblock struct (in the first block), we hide an inode with **index -1** to store the user records.

in the data blocks of the user inode, we record the following:

superuser name: chen
superuser id: 0
password: 09080223

normal user name: dianna
normal user id: 1
password:12345678

C Library

(We implemented all methods except extra credit ones.)

```
struct dirent* current_direct;
// The global variable used to store the current directory
// Help to find a path from ./

struct dirent* root_direct;
// The global variable used to store the
// Help to find a path from the root
```

```
int disk_open(char* diskname);
// The initialization process that should be called at the beginning

int disk_close( );
// The cleanup process

File* f_open(char* filename, char* mode);
// f_open according to the mode
// superuser can access the file in whatever mode it is

int f_read(File *file, void* buffer, int num);
// read a certain number of bytes into buffer

int f_write(File* file, void* buffer, int num);
// write a certain number of bytes into file

int f_close(File* file);
// close the file
// haven't implemented the open file table yet

int f_seek(File* file, int num, int mode);
// allow three modes: SEEK_SET, SEEK_CUR, SEEK_END

int f_rewind(File* file);

void f_path(char* path);
// The helper method that stores the full path of a directory into 'path'

int f_stat(char* filename);

int f_remove(char* filename);
// if the file is not created by the user, it can't be removed

struct dirent* f_opendir(char* directory);
// open the target directory and set its offset -1

struct dirent* f_readdir(struct dirent* directory);
// read through sub-directories of the given directory
// update its offset

int f_closedir(struct dirent* directory);
```

```
// set its offset back to -1

struct dirent* f_mkdir(char* path_name);

int f_rmdir(char* path_name,int flag);
// if flag = 0, remove the directory only if it is empty
// if flag = 1, remove the whole directory recursively
// if the directory is not created by this user, it can't be removed.

int f_changeMod(int inode, int permission);
// if the current user is not the author of the file, the user cannot edit
the permission of the file.

int f_userAuthen(char* username, char* password);
// compare the username and the password that the user has entered with the
info stored in the user inode

void ls_helper(char* content,struct dirent* direct,int flag);
// if flag = 0, only print out the name
// if flag = 1, print out the long information
// if flag = 2, print out with * if it is executable
```

Shell

user authentication:

After the shell is run, we will have a **user authentication**.

The user has to enter the username and the password to verify. If the user is not authenticated, the shell will terminate; if the user is recorded as a super user/normal user, the regular shell will be loaded.

shell functions:

```
ls usage: ls [ /-l/-F] <pathname> <pathname> …
rm usage:   rm <filename>
            rm -rf <pathname> <pathname> …
cat usage:  cat
            cat <filename> <filename> …
more usage: more <filename> <filename> …
mkdir usage: mkdir <pathname> <pathname> …
rmdir usage: rmdir <pathname> <pathname> …
pwd usage: pwd
cd usage: cd <pathname>
chmod
  Usage: chmod [options] mode file...
```

```
  Mode => Symbolic: [3 char] rwx/RWX,
      each char: uppercase=>ALLOW lowercase=>NOTALLOW
  Mode => Absolute: [1 int], the sum of read(4), write(2), execute(1)
```

Redirection:

```
< input.txt  or  <input.txt
> output.txt  or  >output.txt
>> output.txt  or   >>output.txt
```

How to Load the shell:

```
  make
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
  ./format DISK
  ./mysh

To exit:
=> exit
```

How did we test:

C library:

We wrote tests for each library function in **test.c**, and we didn't start to write the shell until we tested all library functions.

Shell:

We designed multiple-layer directories, short text file, long text file, and file that contains a path in the DISK by **format.c**, and use this DISK to test shell functions.

Revision after Demo:

1. (Consistency) We don't have permission settings for directories, only files.
2. (Debug) We now have implemented MORE command, please use a narrow terminal window to test. We have a relatively long text file that can be used to test this command at: root/layerone/layertwo/more.txt
3. (Redirection) We fixed the bugs with redirections and CAT commands. Now, our shell allows using cat >output.txt to create a new file and write content into it. We have also implemented the redirection from UNIX binaries.

Known Limitation

1. Since we will read all data blocks in the disk into the memory at the beginning, the size of the disk cannot be too large.
2. We didn't handle different processes and their corresponding open file tables.

Extra Credit

1. Support "rm -rf"