BetterShell Software Design Document

Bryan Martin, Joseph Listro

3 October 2016

Champlain College

Contents

1 Introduction

This software design document will be an overview of the design choices that our team has made in preparation of implementing a shell for use with a FAT12 file system.

2 How the shell works

The shell runs user commands. When run, it will display a prompt and wait for the user to input a command. When the user has finished their input and pressed enter, the shell will fork off a child process to perform the operations required by the user's command. After the command has been completed, the shell will display any appropriate error messages or output from the process and then display the initial prompt again.

3 Shell Functions

3.1 Concatenate

The concatenate command prints the contents of x to the i/o buffer. If x is a directory or does not exist, then the command will display an error.

3.1.1 Usage

```
$ cat \em{x}
```

3.1.2 Function Prototype

```
int cat();
```

```
int cat(char* target);
```

3.1.3 Function Execution

1. Check that the function was handed exactly one argument.
2. Open the file.
3. Print the contents of the file to the i/o buffer.
4. Print newline character.
5. Function terminates.

3.2 Change Directory

The change directory command will change the shell's working directory to x. If there is not argument specified, then the cd command changes the working directory to the home directory.

3.2.1 Usage

```
$ cd x
```

3.2.2 Function Prototype

```
int cd();
ind cd(char *targetDir);
```

3.2.3 Function Execution

1. Check the number of arguments specified in the function called.

2. If there is no argument specified, change the working directory to the home directory.

3. If there is one argument specified, change the working directory to the specified argument.

4. Function terminates.

3.3 Disk Free

The disk free command will print the number of free logical blocks. Any arguments given to it will be ignored.

3.3.1 Usage

```
$ df
```

3.3.2 Function Prototype

```
int df();
```

Both relative and absolute file names can be provided to this function, a parameter beginning in "/" will be assumed to be absolute, and a parameter beginning with anything else will be assumed to be relative.

3.3.3 Function Execution

1. Print value of free blocks from variable.

3.4 List

The list function will perform three different functions depending on the specified argument. If the argument is the name of a file, then the list function will print the name of the file, the file extension, file type, FLC, and file size. If the name of a directory is the specified argument, then the function will list the names of all the files and sub-directories that are contained in the specified directory in addition to the current and parent directory. Alternatively, if the argument provided is a blank space, then the function will list each entry in the current directory, as well as the current and parent directories, with the file extension for each entry and it's file size in bytes, FLC, and file type. Lastly, if there are two or more arguments, the list function will return an error message.

3.4.1 Usage

```
$ ls x
```

3.4.2 Function Prototype

```
int ls();
int ls(char *target);
```

3.4.3 Function Execution

1. Check there are less than two arguments.
2. Check if the specified argument is a file.
3. If it is, print the name of the file, the file extension, the file size, FLC, and file type.
4. Check if the specified argument is a file directory.

5. If it is, enter the directory and cycle through each entry, printing the file name for each entry.

6. Check if the specified argument is a blank space.

7. If it is, enter the directory and cycle through each entry, including the current and parent directory, printing the file name, file extension, FLC, file size, and file type for each entry.

8. Function terminates.

3.5 Make Directory

The make directory command creates a new directory inside the current (working) directory.

3.5.1 Usage

```
$ mkdir x
```

3.5.2 Function Prototype

```
int mkdir();

int mkdir(char* newDir);
```

3.5.3 Function Execution

1. Check that exactly one argument has been specified.

2. Create a new directory inside the current one.

3. The working directory remains the same.

4. Function terminates.

3.6 Print Working Directory

This function prints the working directory to the I/O buffer.

3.6.1 Usage

```
$ pwd
```

3.6.2 Function Prototype

```
int pwd();
```

3.6.3 Function Execution

1. Ignore any specified arguments.
2. Print the path of the current working directory to the I/O buffer.
3. Function terminates.

3.7 Remove

The remove command will delete a file from the file system.

3.7.1 Usage

```
$ rm x
```

3.7.2 Function Prototype

```
int rm();
```

```
int rm(char *filename);
```

3.7.3 Function Execution

1. Check that exactly one argument has been specified and it is a file name that exists in the working directory.
2. Remove the entry.
3. Optimize the memory usage of the parent directory.
4. Free the data sectors of the target file.
5. Update appropriate data structures.
6. Terminate function.

3.8 Remove Directory

The remove command will delete a directory from the file system.

3.8.1 Usage

```
$ rmdir x
```

3.8.2 Function Prototype

```
int rmdir();
int rmdir(char *dirname);
```

3.8.3 Function Execution

1. Check that exactly one argument has been specified and it is a directory that exists inside the working directory.
2. Remove the entry
3. Optimize the memory usage of the parent directory.
4. Free the data sectors of the target file.
5. Update appropriate data structures.
6. Recursively follow steps 1-6 for all child directories.
7. Terminate function.

3.9 Touch

The touch function will create a new file, or alert the user that a file by that name already exists. Both relative and absolute file names can be provided to this function, a parameter beginning in "/" will be assumed to be absolute, and a parameter beginning with anything else will be assumed to be relative.

3.9.1 Usage

```
$ touch \em{x}
```

3.9.2 Function Prototype

```
$ int touch();
```

```
int touch(char* filename);
```

3.9.3 Function Execution

1. Check argument for a leading "/" (indicates an absolute path)
2. In the FAT, find the directory that contains the file being targeted.
3. Search the directory for files that match the file name given.
4. If not found, add it too the FAT.
5. If found, print that the file already exists.
6. Write to file if necessary.

4 Conclusion

As changes are made to the BetterShell projects, this document will be kept up-to-date, making this an evergreen document.