Robert Bethune
Evan Schipellite
CSI 385 - 02
1 February 2015

FAT Design

Shell Structure

Description:

Runs in a loop, waiting for user input. When the user logs out or types an exit key, the loop will end and the shell will terminate. Anything that the user types will then be processed within the shell. The shell will attempt to validate the command with any of its listed commands, allowing for a child process of that command to be created if valid. Otherwise, the shell will either return a statement noting an incorrect command or it will give a prompt for the proper amount of parameters for the desired command.

Notable Functions & Parameters:

#define INPUT_LENGTH

Provides a base string length for all command processing.

char* m_Command;

char* m_Parameters;

void main();

This prompts the user for their desired command. If exit is entered, the application quits. Otherwise, if the first part of their input line is recognized as a command, the arguments are acquired and the appropriate command is executed alongside the arguments.

void GetArguments(char* commandName, char* commandLIne, char** arguments);

Parses user input following their accepted command. This allows for the creation of a commandInfo string, which contains all required arguments for the execution of commands. The initial string in the character array is also set to the command name, allowing it to be called following the argument acquisition. This utilizes strtok to parse the command line into the character array.

Testing:

- User Input
    - If not a known command
        - Returns to prompt
    - If ‘exit’ is entered
        - Quits the shell
    - If known command
        - Execute command
            - Command will handle error checking on parameters

cat X

Description:

Prints out the current file information to the screen. The user can specify a file path, allowing them to designate whether the file is within a relative or absolute file path.

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Utilized to access byte information from the current directory.

unsigned char rootDirectory[4 * 512 * 14];

Utilized to access byte information for the root directory.

unsigned short initialDirectorySector;

Saves the initial Directory to set back after cd updates the path.

unsigned char initialDirectory[1024];

Saves the initial Directory path to set back after cd updates the path.

unsigned short currentDirectorySector;

Constantly updated to utilize the current relative directory sector.

unsigned short fileInitialPath;

Initial FLC to follow when outputting file contents.

unsigned char* filePath;

Initial file path provided by the user.

void followPath(char* input);

Utilizes cd to temporarily move the current directory examined. The last argument is then used as the directory to remove. If the path is successfully updated, this will then proceed to look for the file. Otherwise, this returns with an error.

void checkFile(unsigned char* fileName);

Checks if the file name exists depending on whether the file is to be created in the root directory or subdirectory.

void checkSubDirectory(unsigned char* fileName);

Searches through the subdirectory, attempting to match the given file name within the directory sector. If the file is found, the command exits with an error message. Otherwise, it follows the FLC and outputs all file contents.

void checkRootDirectory(unsigned char* directoryName);

Searches through the root directory, attempting to match the given file name within the directory sector. If the file is found, the command exits with an error message. Otherwise, it follows the FLC and outputs all file contents.

int outputSubDirectoryFile(unsigned char* file);

Checks if the file can be output within the current subdirectory.

nt outputRootDirectoryFile(unsigned char* file);

Checks if the file can be output within the root directory.

int readSpecificSector(unsigned char* bytes);

For each sub directory sector, this reads to corresponding sector and updates the current directory sector examined.

void readRootSector(unsigned char* bytes);

Reads the entry root sector into a byte array.

Testing:

- If no parameters
    - Error Message: No file specified

- If multiple parameters
    - Error Message: Multiple arguments provided
- If one parameters
    - If file does not exist
        - Error Message: Could not find file
    - If argument is a directory
        - Error Message: Argument is a directory

cd X

Description:

Changes the current directory to the designated directory. If no argument is available, places the user at the home directory.

Notable Functions & Parameters:

unsigned char* rootDirectory[4 * 512 * 14];

If cd begins in the root, this is used to properly find the initial directory sector.

unsigned char* cDirectory[4 * 512];

If cd is not in the root, as the current directory sector will no longer be 0, then this is used to read from the current directory.

int findDirectory(unsigned char* directoryName);

Iterates through the root directory and attempts to match the directory name will the file names found. This also checks to ensure that the directories examined are valid. If found, the current directory and current directory sector is updated, and the function returns a match.

void readRoot(unsigned char* bytes);

Reads the root directory information into a character array.

int readSpecifiedSection(unsigned char* bytes);

Reads the current directory sector, acquiring the appropriate fat entry. This is then used to examine the next FLC.

unsigned char* convertToUpper(unsigned char* userInput);

Converts a char array to upper case.

unsigned char* convertToLower(unsigned char* userInput);

Converts a char array to lower case.

int searchPath(unsigned char* directoryName);

Searches the current directory for the input directory name. This function is called for each string parsed from the pile path, allowing it to progress through the directory until it successfully completes all passes, or until is unable to find the directory.

Testing:

- If floppy cannot be opened
    - Error: could not open the floppy
- If no argument provided
    - Move user to home directory
- If parameters
    - Error message if
        - Directory is invalid
            - No available, accessible, or a file
            - Error: No directory found
        - More than one parameter is provided
            - Error: Multiple arguments entered

df

Description: Prints out the free space on the current disk mounted. If the user provides any parameters, these are ignored.

Notable Functions & Parameters:

Void printDiskFree();

Checks for available free blocks and prints out the free blocks / total blocks.

void readFAT12Table(int sector, unsigned char * fat);

Reads the FAT12Table into a byte array.

Testing:

- Outputs an error message if no disk is mounted
    - Error Message: No disk mounted

ls (X)

Description:

If X is a file, this should list out the file name, extension, type FLC, and size. If X is a directory, this will list out the names of all files within the directory, also detailing file information. If X is empty, this will print out all file information for the current working directory. X can be a relative or absolute path.

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Handles byte reading for subdirectories.

unsigned char rootDirectory[4 * 512 * 14];

Handles byte reading for the root directory.

unsigned short lsDirectory;

Saves a separate short based on the current directory sector. This is designed to avoid overriding the current directory sector.

unsigned char* file;

This references the user input dictating the path to the file or directory to output.

unsigned char* initialDirectory[1024];

A copy of the initial directory. This is set back after completion.

unsigned short initialDirectorySector;

A copy of the initial directory sector. This is set back after completion.

void listFilesInDirectory();

Lists out all file information in the current directory.

void listFilesInRoot();

Lists out all file information in the root directory.

void searchSector();

Prints out all file information for the current sector examined.

void searchRoot();

Prints out all file information for the entire root directory.

int readSpecificSector(unsigned char* bytes);

Reads and sets the lsDirectory to the current sector being examined.

void readRootSector(unsigned char* bytes);

Reads in the entire Root Directory.

void handleInput(unsigned char* bytes);

When a path parameter is given, after the cdCommand moves to the second to last directory, this function handles the final examination. The parameter represents the file or directory name, and the function determines whether it should output the directory contents, or proceed to list a single directory.

int checkIfDirectory(unsigned char* file);

Checks the current directory to see if the file is a directory.

int listSpecificFile(unsigned char* file);

Outputs the file information for a specific file.

void handleSubDirectory(unsigned char* file);

Handles output if the last parameter is a subdirectory.

void handleRootDirectory(unsigned char* file);

Handles output if the last parameter is within the root directory.

int checkIfDirectoryInRoot(unsigned char* file);

Checks if the root directory has a matching file that is a directory.

int listSpecificFileInRoot(unsigned char* file);

Lists a specific file within the root directory.

Testing:

- If no parameters
  - Prints out current directory contents
- If parameters
  - If file
    - Prints out file information
    - If not a valid path to file
      - Error: Could not find file
  - If directory
    - Checks to see absolute / relative path
      - If incorrect path to directory
        - Error: Could not find file
- If multiple parameters
  - Error: Multiple arguments entered

mkdir X

Description:

This will create a new directory with the X name in the current working directory or X path.

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Utilized to access byte information from the current directory.

unsigned char rootDirectory[4 * 512 * 14];

Utilized to access byte information for the root directory.

unsigned short initialDirectorySector;

Saves the initial Directory to set back after cd updates the path.

unsigned char initialDirectory[1024];

Saves the initial Directory path to set back after cd updates the path.

unsigned short currentDirectorySector;

Constantly updated to utilize the current relative directory sector.

unsigned char* filePath;

Initial file path provided by the user.

void followPath(char* input);

Utilizes cd to temporarily move the current directory examined. The last argument is then used as the directory to remove. If the path is successfully updated, this will then proceed to look for the directory. Otherwise, this returns with an error.

void checkDirectory(unsigned char* directoryName);

Checks if the directory name exists depending on whether the file is to be created in the root directory or subdirectory.

void checkSubDirectory(unsigned char* directoryName);

Searches through the subdirectory, attempting to match the given directory name within the directory sector. If the directory is found, the command exits with an error message. Otherwise, it continues to createFile();

void checkRootDirectory(unsigned char* directoryName);

Searches through the root directory, attempting to match the given directory name within the directory sector. If the directory is found, the command exits with an error message. Otherwise, it continues to createFile();

int checkDirectoryInSub(unsigned char* directoryName);

Checks if the directory exists within the current subdirectory.

nt checDirectoryInRoot(unsigned char* directoryName);

Checks if the directory exits within the root directory.

int readSpecificSector(unsigned char* bytes);

For each sub directory sector, this reads to corresponding sector and updates the current directory sector examined.

void readRootSector(unsigned char* bytes);

Reads the entry root sector into a byte array.

void readFAT12Table(int sector, unsigned char * fat);

Reads the FAT12Table into a byte array.

void writeFAT12Table(unsigned char * fat);

Writes the FAT12Table entries into a byte array.

void writeRootSector();

Writes the root sector contents to save to the floppy.

void writeSubSector(int initialSector);

Writes the SubSector contents to save to the floppy.

void createFile(unsigned char* file);

Creates the directory within the current directory. This first builds the directory template, setting the directory name, extension, FLC, and file size. This also adjusts the attributes to reflect the directory status. Then, depending on whether the root or subdirectory must be edited, this writes it to the corresponding sector and updates the FAT entry. The '.' and '..' directories are also created and linked to the specified directory.

Testing:

- If one parameter
    - If X exists
        - Error: Directory already exists
    - If X does not exist
        - Check relative / absolute path
            - Create directory X at path
                - If directoryName too long
                    - Shortens to 8
                - If no entries available
                    - Error: No memory space available
- If no parameters
    - Error: No directory specified

- If multiple parameters
    - Error: Multiple arguments provided

pwd

Description: Prints out the absolute path of the current directory. This essentially just accesses the shared memory, locates the variable currentDirectory, and then outputs the result. This information is set and adjusted by other commands.

Testing:

rm X

Description: Attempts to remove the specified file, if it exists. Can have an absolute or relative path.

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Utilized to access byte information from the current directory.

unsigned char rootDirectory[4 * 512 * 14];

Utilized to access byte information for the root directory.

unsigned short initialDirectorySector;

Saves the initial Directory to set back after cd updates the path.

unsigned char initialDirectory[1024];

Saves the initial Directory path to set back after cd updates the path.

unsigned short currentDirectorySector;

Constantly updated to utilize the current relative directory sector.

unsigned char* filePath;

Initial file path provided by the user.

void followPath(char* input);

Utilizes cd to temporarily move the current directory examined. The last argument is then used as the remove file. If the path is successfully updated, this will then proceed to look for the file. Otherwise, this returns with an error.

void checkFile(unsigned char* file);

Depending on whether the current directory sector is in the subdirectory or root directory, this will proceed to the appropriate function.

void checkSubDirectory(unsigned char* file);

Searches through the subdirectory, attempting to match the given file name within the directory sector. If the file is found, this proceeds to remove the file and write the sector.

void checkRootDirectory(unsigned char* file);

Searches through the root directory, attempting to match the given file name. If the file is found, this removes the file and writes back to the root sector.

int removeSubDirectoryFile(unsigned char* file);

Searches through the sector, matching the input file name against all entries. This also tests attributes to ensure the file exists, and it is not a directory. This then proceeds to remove the file based on its flc.

int removeRootDirectoryFile(unsigned char* file);

Searches through the root directory, matching the input file name against all entries. This also tests attributes to ensure the file exists, and it is not a directory. This then proceeds to remove the file based on its flc.

int readSpecificSector(unsigned char* bytes);

For each sub directory sector, this reads to corresponding sector and updates the current directory sector examined.

void readRootSector(unsigned char* bytes);

Reads the entry root sector into a byte array.

void removeFile(short startFLC);

Based on the startFLC, this removes the file and all related cluster connections. This will proceed to locate the nextLC, erase the data entry, and set the LC to 0xfff. This is written to the current sector, and later the contents are updated within the FAT12Table.

void readFAT12Table(int sector, unsigned char * fat);

Reads the FAT12Table into a byte array.

void writeFAT12Table(unsigned char * fat);

Writes the FAT12Table entries into a byte array.

void writeRootSector();

Writes the root sector contents to save to the floppy.

void writeSubSector(int initialSector);

Writes the SubSector contents to save to the floppy.

Testing:

- If no parameters
  - Error Message: No file specified
- If multiple parameters
  - Error Message: Multiple arguments provided
- If one parameters
  - If file does not exist
    - Error Message: Could not find file
  - If argument is a directory
    - Error Message: Argument is a directory

rmdir X

Description: Removes the provided directory from the disk. Can have an absolute or relative path.

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Utilized to access byte information from the current directory.

unsigned char rootDirectory[4 * 512 * 14];

Utilized to access byte information for the root directory.

unsigned short initialDirectorySector;

Saves the initial Directory to set back after cd updates the path.

unsigned char initialDirectory[1024];

Saves the initial Directory path to set back after cd updates the path.

unsigned short currentDirectorySector;

Constantly updated to utilize the current relative directory sector.

unsigned char* filePath;

Initial file path provided by the user.

void followPath(char* input);

Utilizes cd to temporarily move the current directory examined. The last argument is then used as the directory to remove. If the path is successfully updated, this will then proceed to look for the directory. Otherwise, this returns with an error.

void checkDirectory(unsigned char* directoryName);

Depending on whether the current directory sector is in the subdirectory or root directory, this will proceed to the appropriate function.

void checkSubDirectory(unsigned char* directoryName);

Searches through the subdirectory, attempting to match the given directory name within the directory sector. If the directory is found, this proceeds to remove the directory and write the sector.

void checkRootDirectory(unsigned char* directoryName);

Searches through the root directory, attempting to match the given directory name. If the directory is found, this removes the directory and writes back to the root sector.

int removeSubDirectory(unsigned char* directoryName);

Searches through the sector, matching the input directory name against all entries. This also tests attributes to ensure the directory exists, it is empty, and it is not a file. This then proceeds to remove the directory based on its flc.

int removeRootDirectory(unsigned char* directoryName);

Searches through the root directory, matching the input directory name against all entries. This also tests attributes to ensure the directory exists, it is empty, and it is not a file. This then proceeds to remove the directory based on its flc.

int readSpecificSector(unsigned char* bytes);

For each sub directory sector, this reads to corresponding sector and updates the current directory sector examined.

void readRootSector(unsigned char* bytes);

Reads the entry root sector into a byte array.

void removeDirectory(short startFLC);

Based on the startFLC, this removes the directory and all related cluster connections. This will proceed to locate the nextLC, erase the data entry, and set the LC to 0xfff. This is written to the current sector, and later the contents are updated within the FAT12Table.

void readFAT12Table(int sector, unsigned char * fat);

Reads the FAT12Table into a byte array.

void writeFAT12Table(unsigned char * fat);

Writes the FAT12Table entries into a byte array.

void writeRootSector();

Writes the root sector contents to save to the floppy.

void writeSubSector(int initialSector);

Writes the SubSector contents to save to the floppy.

int checkIfDirectoryIsEmpty(int flc);

Checks to see if the directory is empty. This occurs if the directory only contains '.' and '..' as entries.

<u>Testing:</u>

- If no parameters
  - Error Message: No directory specified
- If multiple parameters
  - Error Message: Multiple arguments provided
- If one parameters
  - If directory does not exist
    - Error Message: Directory does not exist
  - If argument is a file

        - Error Message: Argument is a directory
    - If directory is not empty
        - Error Message: Directory is not empty

touch X

Description:

This function will look at X and see whether or not it exists already, if it does it will return that it does. If it does not exist it will look and try to create the new file or directory in either the name or file path location

Notable Functions & Parameters:

unsigned char cDirectory[4 * 512];

Utilized to access byte information from the current directory.

unsigned char rootDirectory[4 * 512 * 14];

Utilized to access byte information for the root directory.

unsigned short initialDirectorySector;

Saves the initial Directory to set back after cd updates the path.

unsigned char initialDirectory[1024];

Saves the initial Directory path to set back after cd updates the path.

unsigned short currentDirectorySector;

Constantly updated to utilize the current relative directory sector.

unsigned char* filePath;

Initial file path provided by the user.

void followPath(char* input);

Utilizes cd to temporarily move the current directory examined. The last argument is then used as the directory to remove. If the path is successfully updated, this will then proceed to look for the file. Otherwise, this returns with an error.

void checkFile(unsigned char* fileName);

Checks if the file name exists depending on whether the file is to be created in the root directory or subdirectory.

void checkSubDirectory(unsigned char* fileName);

Searches through the subdirectory, attempting to match the given file name within the directory sector. If the file is found, the command exits with an error message. Otherwise, it continues to createFile();

void checkRootDirectory(unsigned char* directoryName);

Searches through the root directory, attempting to match the given file name within the directory sector. If the file is found, the command exits with an error message. Otherwise, it continues to createFile();

int checkSubDirectoryFileExists(unsigned char* file);

Checks if the file exists within the current subdirectory.

nt checkRootDirectoryFileExists(unsigned char* file);

Checks if the file exits within the root directory.

int readSpecificSector(unsigned char* bytes);

For each sub directory sector, this reads to corresponding sector and updates the current directory sector examined.

void readRootSector(unsigned char* bytes);

Reads the entry root sector into a byte array.

void readFAT12Table(int sector, unsigned char * fat);

Reads the FAT12Table into a byte array.

void writeFAT12Table(unsigned char * fat);

Writes the FAT12Table entries into a byte array.

void writeRootSector();

Writes the root sector contents to save to the floppy.

void writeSubSector(int initialSector);

Writes the SubSector contents to save to the floppy.

void createFile(unsigned char* file);

Creates the file within the current directory. This first builds the file template, setting the filename, extension, FLC, and file size. Then, depending on whether the root or subdirectory must be edited, this writes it to the corresponding sector and updates the FAT entry.

Testing:

- If one parameter
  - If X exists
    - Error: File already exists
  - If X does not exist
    - Check relative / absolute path
      - Create file X at path
        - If filename too long
          - Shortens to 8
        - If no entries available
          - Error: No memory space available
- If no parameters
  - Error: No file specified
- If multiple parameters
  - Error: Multiple arguments provided