**File System**

**ECE 4310**

**Programming Assignment Report**

**Sergio R. Minera**

**May 2, 2019**

File System

**Objective:**

The objective of this assignment was to develop a program that implemented a file system which would allow the user to create, delete, read and write files into a directory by using a File allocation table and thus allow the student developing this code to understand more efficiently how a File directory uses the file allocation table to store addresses of the data space and create file within the directory that will store data into the data space based on the addresses stored into the FAT.

**Source Code:**

// CPPfileSystem.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include "FileDirectory.h"

using namespace std;

int \_tmain(int argc, \_TCHAR\* argv[])

{

FileDirectory fileDirectory;

char data[1024];

// Write a driver function, i.e.main(), to test this program by

//1. create and write a file, file1, of 40 bytes,

fileDirectory.write("file1", 40, data, 2019, 4, 16, 15, 29, 30);

fileDirectory.printData("file1");

//2. create and write a file, file2, of 200 bytes,

fileDirectory.write("file2", 200, data, 2019, 4, 16, 15, 50, 30);

//3. create and write a file, file3, of 300 bytes,

fileDirectory.write("file3", 300, data, 2019, 4, 16, 16, 35, 30);

//4. create and write a file, file4, of 500 bytes.

fileDirectory.write("file4", 500, data, 2019, 4, 16, 17, 35, 30);

fileDirectory.printDirectory();

//5. delete file2,

fileDirectory.deleteFile("file2");

//6. create and write a file, file4, of 500 bytes.

fileDirectory.write("file4", 500, data, 2019, 4, 16, 17, 35, 30);

fileDirectory.printDirectory();

return 0;

}

class FileDirectory

{

private:

unsigned char fileDirectory[4][32]; /\* a max of N file directories. Each has 32 bytes.

File Name[7:0] : 8-byte

File Name Extension[8:10]: 3-byte

Date[25:24] and Time[23:22] of file creating or modification

16-bit First sector address[27:26]

32-bit File size[31:28], for a file size up to 4 Gbytes

The remaining bytes are for other File attributes and future expansion. \*/

unsigned short int FAT16[256]; //256 clusters only

unsigned char Data[1024]; //256 clusters \* 4 bytes/cluster = 1024 bytes.

public:

FileDirectory(); //: to initialize all entries in the fileDirectory and FAT16 to be 0; i.e.safe values.

bool create(char filename[], int numberBytes);

bool deleteFile(char filename[]);

bool read(char filename[]);

bool write(char filename[], int numberBytes, char data[], int year, int month, int day, int hour, int minute, int second);

void printClusters(char filename[]);

void printDirectory();// prints all the files of the directory.

void printData(char filename[]);// prints the data of a file.

};

#include "stdafx.h"

#include "FileDirectory.h"

#include <iostream>

#include <string>

#define EOFile 0xffff

using namespace std;

FileDirectory::FileDirectory()

{

//purpose : to initialize all entries in the fileDirectory and FAT16 to be 0; i.e.safe values.

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 32; j++) fileDirectory[i][j] = 0;

}

for (int i = 0; i < 256; i++) FAT16[i] = 0;

for (int i = 0; i < 1024; i++) Data[i] = 0;

}

bool FileDirectory::create(char filename[], int numberBytes)

{

// (1) to check if there is an unused entry in the File Directory; (i.e.the first character of the file name in the File Directory is zero).Return false if not true.

bool flagEmpty = false;

bool enoughSpace = false;

int freeclusters = 0;

for (int i = 0; i < 4; i++)

{

if (fileDirectory[i][0] == 0)

{

flagEmpty = true;

for (int index = 1; index < 255; index++)

{

if (FAT16[index] == 0 || FAT16[index] == 1) freeclusters ++;

}

if (freeclusters \* 4 >= numberBytes)

{

enoughSpace = true;

break;

}

}

}

// (2) to check if there are enough unused clusters to store the file with the numberBytes. Return false if not true.

if (enoughSpace == false)

{

cout << "Not enough space available in FAT16" << endl;

return false;

}

else if (flagEmpty == false)

{

cout << "Not enough space available in File Directory" << endl;

return false;

}

else return true;

}

bool FileDirectory::deleteFile(char filename[])

{

// (0) to check if the file to be deleted; filename[]; is in the Directory.If not; return false.

int i, j;

for (i = 0; i < 4; i++){

for (j = 0; j < 8; j++)

{

if (fileDirectory[i][j] != filename[j])break;

}

if (j == 8) break;

}

if (i == 4) return false;

// (1) to change the first character of the file name in the File Directory to be zero;

fileDirectory[i][0] = NULL;

// (2) to change all entries of the clusters of this file in the FAT to 1; i.e.; unused.

int firstSectorAddress = (fileDirectory[i][27] << 8) + fileDirectory[i][26];

int sectors[256];

//Initialize an empty array (-1 value) to store sectors

for (int i = 0; i < 256; i++)

{

sectors[i] = -1;

}

//Use nextSector counter to make sure sectors are linked together in sectors array

int nextSector = 1;

sectors[0] = firstSectorAddress;

for (i = firstSectorAddress; FAT16[i] != EOFile; i++)

{

sectors[nextSector] = FAT16[i];

nextSector++;

}

//Use sectors array to delete entries in FAT16

for (int j = 0; sectors[j] != -1; j++)

{

FAT16[sectors[j]] = 1;

}

}

bool FileDirectory::read(char filename[])

{

//purpose: to read of data from data[] array from the file with the specified file name.

//(0) to check if the file to be read filename[], is in the Directory.If not; return false.

int i, j;

for (i = 0; i < 4; i++){

for ( j = 0; j < 8; j++)

{

if (fileDirectory[i][j] != filename[j])break;

}

if (j == 8) break;

}

//3.2 if not found, return false,

if (fileDirectory[i][0] == 0) return false;

if (i == 4) return false;

//(1) use the file name to get the file information from the File Directory; including date; time; number of bytes and the first cluster address;

int year, month, day, hour, minute, second, date, time;

date = (fileDirectory[i][25] << 8) + fileDirectory[i][24];

year = 1980 + (date >> 9);

month = (date & 0x07e0 )>> 5;//

day = date & 0x001f ; //5 LSB

time = (fileDirectory[i][24] << 8) + fileDirectory[i][23];

hour = time>>11 ; minute = (time & 0x07e0)>>5; second = (time&0x001f) <<1 ;

//Initialize an empty array (-1 value) to store sectors

int sectors[256];

for (int i = 0; i < 256; i++)

{

sectors[i] = -1;

}

// (2) use the first cluster address to get all the cluster addresses of this file from the FAT - 16;

int firstSectorAddress = (fileDirectory[i][27] << 8) + fileDirectory[i][26];

sectors[0] = firstSectorAddress;

//(3) get all the remaining cluster addresses from the FAT to read the data from the disk / flash memory.

//Use nextSector counter to make sure sectors are linked together in sectors array

int nextSector = 1;

for (i = firstSectorAddress; FAT16[i] != EOFile; i++)

{

sectors[nextSector] = FAT16[i];

nextSector++;

}

return true;

}

bool FileDirectory::write(char filename[], int numberBytes, char data[], int year, int month, int day, int hour, int minute, int second)

{

if (create(filename, numberBytes) == false) return false;

//purpose: to write numberBytes bytes of data from data[] array into the file with the specified file name

else

{

int sectors[256];

for (int i = 0; i < 256; i++)

{

sectors[i] = -1;

}

//(0) to look for the first unused entry(0 or 1) in the FAT - 16, and use it as the First Cluster Address.

int firstSectorAddress = 0;

for (int i = 2; i < 255; i++)

{

if (FAT16[i] == 0 || FAT16[i] == 1)

{

firstSectorAddress = i;

break;

}

}

//(1) to look for the next unused entry(0 or 1) in the FAT - 16, and use it as the Next Cluster Address, and write its value into the FAT - 16.

sectors[0] = firstSectorAddress;

//Use nextSector counter to make sure sectors are linked together in sectors array

//Use lastSector counter to write EOFile into last sector of file

int nextsector = 1;

int lastSector = firstSectorAddress;

for (int i = firstSectorAddress; nextsector - 1 < numberBytes / 4; i++)

{

if (FAT16[i] == 0 || FAT16[i] == 1)

{

sectors[nextsector] = i;

lastSector = sectors[nextsector];

nextsector++;

}

}

//(2) Repeat Step 2 until all clusters are found and the FAT - 16 is updated.

for (int i = 0; i < 256; i++)

{

if (sectors[i] != lastSector)

{

FAT16[sectors[i]] = sectors[i+1];

}

else if (sectors[i] == lastSector)

{

FAT16[sectors[i]] = EOFile;

}

}

//(3) to write / update the file name, extension, date, time, file length and first cluster address into the first unused entry in the File Directory,

//3.1 look for an unused entry in the directory

int i;

for (i = 0; i < 4; i++) {

if (fileDirectory[i][0] == 0)break;

}

//3.2 if not found, return false,

if (i == 4) return false;

//3.3 if found, write all file info into the entry

//write file name

char fname[8];

for (int j = 0; j < 8; j++)

{

fileDirectory[i][j] = filename[j];

fname[j] = filename[j];

}

string file(fname);

//write date

int date;

date = ((year - 1980) << 9) + (month << 5) + day;

fileDirectory[i][24] = date & 0x00FF;

fileDirectory[i][25] = date >> 8;

//write time

int time;

time = (hour << 11) + (minute << 5) + second / 2;

fileDirectory[i][22] = time & 0x00FF;

fileDirectory[i][23] = time >> 8;

//write file length

for (int k = 28; k < 32; k++)

{

fileDirectory[i][k] = numberBytes;

}

//write first sector address

fileDirectory[i][26] = firstSectorAddress & 0x00FF;

fileDirectory[i][27] = (firstSectorAddress >> 8) & 0x00FF;

//write data according to file name for testing dataprint function

for (int i = 0; i < numberBytes/4 + 1; i++)

{

for (int j = 0; j < 4; j++)

{

if (file == "file1") Data[sectors[i] \* 4 - 6 + j] = 'a';

else if (file == "file2") Data[sectors[i] \* 4 - 6 + j] = 'b';

else if (file == "file3") Data[sectors[i] \* 4 - 6 + j] = 'c';

else if (file == "file4") Data[sectors[i] \* 4 - 6 + j] = 'd';

}

}

//retun true.

return true;

}

}

void FileDirectory::printClusters(char filename[])

{

// purpose : to print all the clusters of a file.

// (1) to check if the file to be printed, filename[], is in the Directory.If not, return false.

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 8; j++)

{

if (fileDirectory[i][j] != filename[j]) break;

}

if (j == 8) break;

}

//(2) use the file name to get the file information from the File Directory, including the first cluster address,

int sectors[256];

for (int i = 0; i < 256; i++)

{

sectors[i] = -1;

}

int firstSectorAddress = (fileDirectory[i][27] << 8) + fileDirectory[i][26];

//Find first sector address of the next file in file directory assuring that the clusters that are printed

//are pertaining to its file and not any subsequent file in the directory

int nextFileFirstSectorAddress = (fileDirectory[(i+1) % 4][27] << 8) + fileDirectory[(i+1) %4][26];

sectors[0] = firstSectorAddress;

int nextSector = 1;

for (i = firstSectorAddress; FAT16[i] != EOFile; i++)

{

if (i == nextFileFirstSectorAddress && i != firstSectorAddress)

{

for (i = nextFileFirstSectorAddress; FAT16[i] != EOFile; i++) {}

i++;

}

sectors[nextSector] = FAT16[i];

nextSector++;

}

//(3) use the first cluster address to get all cluster addresses from the FAT - 16,

for (i = 0; sectors[i] != -1; i++)

{

if (sectors[i + 1] == -1)

{

cout << sectors[i] << endl;

}

else cout << sectors[i] << "->";

}

}

void FileDirectory::printDirectory()

{

//purpose: prints all the files of the directory.

//(1) use the file name to get the file information from the File Directory, including the first cluster address

char fileName[8];

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 8; j++)

{

fileName[j] = fileDirectory[i][j];

}

string fName(fileName);

if (read(fileName) == true)

{

cout << fName << " ";

cout << " . ";

int year, month, day, hour, minute, second, date, time;

date = (fileDirectory[i][25] << 8) + fileDirectory[i][24];

year = 1980 + (date >> 9);

month = (date & 0x01e0) >> 5;//

day = date & 0x001f; //5 LSB

time = (fileDirectory[i][23] << 8) + fileDirectory[i][22];

hour = time >> 11;

minute = (time & 0x07e0) >> 5;

second = (time & 0x001f) << 1;

cout << month << "/" << day << "/" << year << " ";

cout << hour << ":" << minute << ":" << second << " ";

int firstSectorAddress = (fileDirectory[i][27] << 8) + fileDirectory[i][26];

int size = fileDirectory[i][28];

cout << size << " ";

cout << firstSectorAddress << endl << endl;

printClusters(fileName);

cout << endl;

}

}

//(2) use the first cluster address to get all cluster addresses from the FAT - 16,

cout << endl;

}

void FileDirectory::printData(char filename[])

{

//purpose: to print the data of a file.

// (1) use the file name to get the file information from the File Directory, including the first cluster address,

int i, j;

for (i = 0; i < 4; i++)

{

for (j = 0; j < 8; j++)

{

if (fileDirectory[i][j] != filename[j]) break;

}

if (j == 8) break;

}

int firstSectorAddress = (fileDirectory[i][27] << 8) + fileDirectory[i][26];

int size = fileDirectory[i][28];

// (2) use the first cluster address to get all cluster addresses from the FAT - 16,

int sectors[256];

for (int i = 0; i < 256; i++)

{

sectors[i] = -1;

}

int nextFileFirstSectorAddress = (fileDirectory[(i+1) % 4][27] << 8) + fileDirectory[(i+1) % 4][26];

sectors[0] = firstSectorAddress;

int nextSector = 1;

for (i = firstSectorAddress; FAT16[i] != EOFile; i++)

{

if (i == nextFileFirstSectorAddress && i != firstSectorAddress)

{

for (i = nextFileFirstSectorAddress; FAT16[i] != EOFile; i++) {}

i++;

}

sectors[nextSector] = FAT16[i];

nextSector++;

}

// (3) use cluster address to read the data of the file.Use the file length to print these data in hexadecimal format.

// For each cluster, the function must output 4 bytes from the data array, thus the math ensures that all 4 bytes are outputed

for (int i = 0; i < size/4; i++)

{

for (int j = 0; j < 4; j++)

{

cout << Data[sectors[i] \* 4 - 6 + j] << " ";

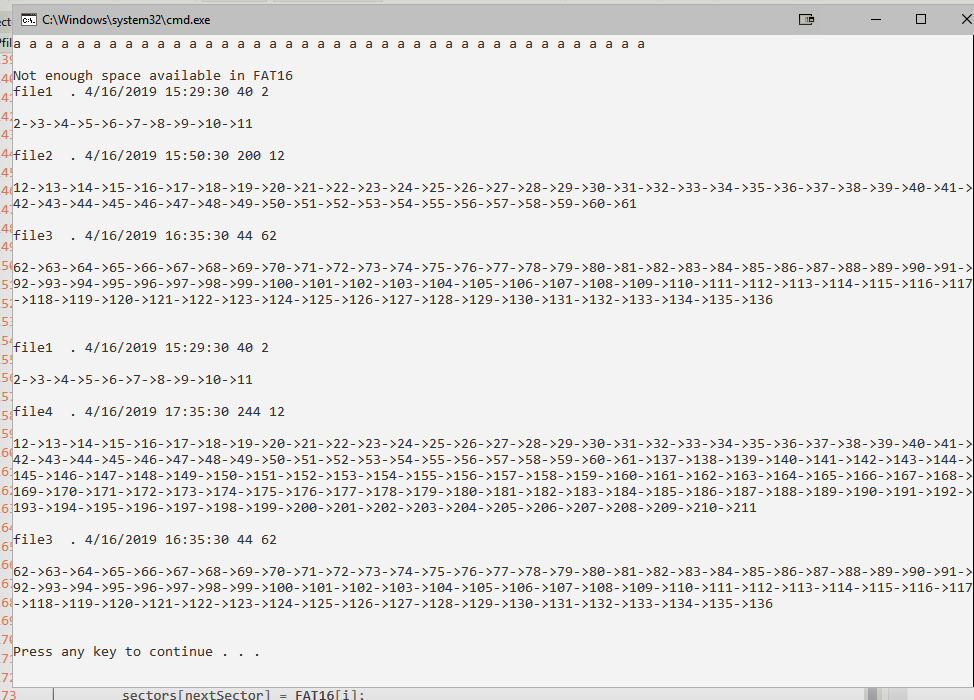
}

}

cout << endl << endl;

}

**Test Results:**



1. .
   1. For the output shown above, the string of ‘a’ characters is the output of printData function, because the write function will assign a character to a file name, all data sectors of that file will have the same character, the print function then proceeds to output values form the data array pertaining to the addresses stored in the FAT table. Because this output is pertaining to file1, it should only output the addresses that store the value of ‘a’ and because its size is 40 bytes, it outputs a total of 40 ‘a’ characters.

Following this file1 is created and its clusters are outputted, similarly file2 and file 3 are created afterward and outputted.

After all three files have been printed, in the main function one can see that file 4 was created as well, however it did not output because the create function returned a false flag due to insufficient space available in the FAT16 table, thus not allowing the write function to write file4 into the directory. However, after file2 is deleted, new clusters are available and thus the printDirectory function is able to output file1, file4 and file3. One thing to note is that because file2 was deleted from the file directory prior to writing file4, file4 is created into the file directory in the space where file2 was stored previously, thus, due to the printDirectory being a sequential print and not sorted in file name, file4 is printed before file3 due to their index values in the file directory.

* 1. Testing conditions:
     1. If you write a file with a length greater than the available disk space you will get a false flag due to insufficient space and a message indicating no space.
     2. If you write a file with the same name the function create that is called within the write file will only check if there is enough space in the directory and in the data space, if both conditions are true then it will just write that file in another index of the file directory, however, code should be expanded to instead prompt user if they desire to overwrite the existing file with the new one or rename the new file.
     3. The create function will flag both bool conditions within it that check for enough space or enough data space, if either of them are false, the function will not write anything.