UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Department of Electronic & Computer Engineering

DATA Forensics Assignment

(Phase 1 & 2)

Student Name: **Dongao Li, Hang Liu**

Student ID:

Supervisor: **D.Heffernan**

Course: **EE6012: DATA Forensics**

November 2017

Requirements

The aim of this assignment is writing a program as a forensic tool to learn the disk partition scheme and FAT, NTFS file systems in detail. First of all, the number of partitions on the disk and each partitions detail information needed to display, like the start sector, the size of partition and file system type.

And then, the information of the first partition that will always be formatted as a FAT-16 need be shown in the tool, which includes the number of sector per cluster, the size of the FAT area, the size of Root Directory and the sector address of cluster#2. In addition, this tool can find the first deleted file on the volume's root directory, and display the name and size of that file, and the number of the first cluster. Also, the first 16 characters of the content of that file need to be shown.

Finally, the details of NTFS volume also require being shown in the tool, such as the number of bytes per sector, the number of sectors per cluster, the sector address of $MFT file record and the type and length of the first two attributes in the $MFT record.

Description of solution

The programming language of this assignment is C++. And the environment of running is Windows or MAC. The first partition just needs read the MBR partition information, because each partitions information are stored in here. We open the file in binary mode using open () and use the seekg () and read () to locate the MBR start address and read it. And every partition table entry has 16 bytes, and the layout of the table is fixed. So we can get the detail information of each partition from those tables. And using the partition type codes table, we could get the type of each partition.

In the FAT-16 volume, the solution is same with the partition of MBR. First of all, we need to seek the address of FAT-16 volume with the information of FAT-16 on the MBR. And according to the layout of the FAT-16 volumes boot sector to get the basic information of this volume and calculate the cluster#2 sector address on the disk. Due to the first byte that is 0xE5 we could find the deleted file in this volume. There is an attention in finding the address of deleted file. Because the starting cluster of files is cluster#2, the starting cluster that gets from directory entry structure needs to subtract two. And then we can calculate the start address using the sector address for cluster#2, cluster number and the number of sector per cluster.

But there is a big problem in reading the data that more than one byte. Because the storage mode of the data is little endian, we need to do the shift operating to get the big endian data. At first, we use the MAKEWORD() function to do the

shift operating. But this function only can be used with the windows library function, which means it is not a standard library function. In order to run it in the other environment, we write a new function to do this shift operating. We divide it into two situations. One is the data we need read from disk is two bytes, that just need to change the order of this two bytes. Other is four bytes, that need to make the third byte and forth byte left shift 16 bits and 24 bits. And the second byte needs to left shift 8 bits, then the four bytes need to do OR operating. The final number is that we need. And the code is following:

```
int getOffset(char arrary[], int start, int num){
    int offset;
    if (num==2){
        offset = ((int)(unsigned char)(arrary[start+1])<<8) | ((int)(unsigned char)arrary[start] & 0x00ff);}
    else if (num==4){
        offset=((int)(unsigned char)arrary[start+1]<< 8 );
        offset= offset | ((int)(unsigned char)arrary[start] & 0x000000ff);
        offset= (offset & 0x0000ffff) | ((int)(unsigned char)arrary[start+3] << 24 ) | ((int)(unsigned char)arrary[start+2] << 16 ) ;}
    return offset;}
```

As for NTFS volume, the function of operation is same with before. But the structure of the NTFS is different with the FAT-16. We just need to seek and read the BPB and extend BPB that both of all are stored in the NTFS boot sector, and we can get all information that we need. But the problem is that the logical cluster number for file $MFT is a longlong type, that mean the function wrote to do shift operating is not suitable. Because this data has eight bytes, the shift operating is very complex. So we decide to use the stringstream type that included in sstream library which is c++ standard library. It can accept different types at the same time. So we put the data to this variable in inverted sequence first, and then put this variable into long int variable to change the data from char to long int. The aim of this operation is getting big endian data. And the code is following:

```
stringstream addr;
for(int i=7; i>=0; i--){
    addr<<(int)(unsigned char)NTFS[0x30+i];}
long int ad;
addr>>ad;
long sec_address = ad * sec_per_cluster;
```

But we found this function also cannot solve this problem, because the element of NTFS array will be stored in the stringstream addr with decimal rather than hexadecimal. That mean the final result of big endian data will be changed. So

we decided to use the original function that does the shift operating directly. The code is following:

```
long long addr;
for(int i=7; i>=0; i--){
    if (i==7){
        addr=(int)(unsigned char)NTFS[0x30+i];
    }else{
        addr= addr<<8 | ((int)(unsigned char)NTFS[0x30+i]    & 0x000000ff) ;
    }
}
```

Testing and results

In this step, we use two image file to test this tool, one is given, and one is created by ourselves. The screen shot of the given image file is following:

```
Please input the path of Image file:
D:\SampleImage\Sample_1.dd
-------------------------------------------------------
~~~~~~~~~~~~~~~~The MBR information~~~~~~~~~~~~~~~~
Partition: 0  Type: FAT-16  Start: 63  Size: 514017
Partition: 1  Type: FAT-32  Start: 578340  Size: 1028160
Partition: 2  Type: NTFS  Start: 1606500  Size: 369495
Partition: 3  Type: NOT-VALID  Start: 0  Size: 0

Total number of valid partitions is: 3

~~~~~~~~~~~~~~~~~~FAT-16 information~~~~~~~~~~~~~~~~~
+++Basic info+++
number of sectors per cluster: 8
The size of the FAT area: 502
The size of the Root directory: 32
The sector address of Cluster #2: 599
++++++++++++++++++++++++++++++++++++++++
The information of first deleted file on the volume's root directory:
The name of the file: ♦OOK TXT
the size of the file: 61134 (eece)
the number of the first cluster: 19
The address of the cluster sector: 735
The first 16 characters of the content of this file:
....Section.A:..

~~~~~~~~~~~~~~~~~~NTFS information~~~~~~~~~~~~~~~~~
The bytes per sector for this NTFS volume: 512
The sectors per cluster for this NTFS volume: 8
The sector address for the $MFT file record: 32
The type and length of the first attribute in the $MFT record:
type:  16        length:  96
The type and length of the second attribute in the $MFT record:
type:  48        length:  104

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
please input q to exit this program!
```

Figure 1 testing one

And the final result screen shot of the Image file created by ourselves, that include only one NTFS partition and two .txt file in this partition, is that:

```
Please input the path of Image file:
D:\SampleImage\xxx.dd
------------------------------------------------------------
~~~~~~~~~~~~~~~~~~~~~~~~The MBR information~~~~~~~~~~~~~~~~~~~~~~~~
Partition: 0  Type: NTFS  Start: 7496  Size: 3801784
Partition: 1  Type: NOT-VALID  Start: 0  Size: 0
Partition: 2  Type: NOT-VALID  Start: 0  Size: 0
Partition: 3  Type: NOT-VALID  Start: 0  Size: 0

Total number of valid partitions is: 1

~~~~~~~~~~~~~~~~~~~~~~~~NTFS information~~~~~~~~~~~~~~~~~~~~~~~~
The bytes per sector for this NTFS volume: 512
The sectors per cluster for this NTFS volume: 8
The sector address for the $MFT file record: 1267256
The type and length of the first attribute in the $MFT record:
type:  16        length:  96
The type and length of the second attribute in the $MFT record:
type:  48        length:  104
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
There is no FAt-16 file system!

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
please input q to exit this program!
q
```

Instructions

We use the eclipse to program and compile this tool. But the debug can be run in the windows environment directly.

Statement of completion

We have completed all of the requirements. And in this assignment, Hang Liu has completed the code of reading the MBR and the FAT-16 partition. Dongao Li has completed the code of reading the NTFS partition, the testing and the final document.

Source code

```
//============================================================================
// Name        : DataForensics.cpp
// Author      : Dongao Li, Hang Liu
// Time        : 23/2/2017
//============================================================================
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string.h>
#include <iomanip>
using namespace std;
struct Partition{ string type; int start; int size;} parts[4];
// the struct of partition that is used to store the information of every partition
ifstream file;

/**
 * @param: char File_path[]
 *
 * @brief: open the Imager file
 *
 * @return: void
 */
void readFile(char File_path[]){
    file.open(File_path,ios::in | ios::binary);
        if (!file.is_open()){
            cout<< "Error opening file!!!"<<endl;
            exit(1);
        }
}

/**
 * @param: char array[], int start, int num
 *
 * @brief: the array[] is the objective array, the start is the
 *         location of element that need to do shift operation
 *         in the array, the num is the number of element that need be operated.
 *         According those three parameters to get the elements
 *         and do the left shift operation.
 *
 * @return: offset
 */
int getOffset(char array[], int start, int num){
```

```
    int offset;
    if (num==2){
        offset = ((int)(unsigned char)(array[start+1])<<8) | ((int)(unsigned
char)array[start] & 0x00ff);
    }
    else if (num==4){
        offset=((int)(unsigned char)array[start+1]<< 8 );
        offset= offset | ((int)(unsigned char)array[start] & 0x000000ff);
        offset= (offset & 0x0000ffff) | ((int)(unsigned char)array[start+3] <<
24 ) | ((int)(unsigned char)array[start+2] << 16 ) ;
    }
    return offset;
}

/**
 * @param: void
 *
 * @brief: read the MBR partition, and put the every partitions
 *         information into parts[4] struct
 *         final to print all of the information
 *
 * @return: void
 */
void readMBR(){
    char buf[64]; // read 64 bytes data
    int validParts=4;
    file.seekg(0x1BE, ios::beg);
    file.read(buf,64);
    cout<<"~~~~~~~~~~~~~~~~~~~~The MBRinformation~~~~~~~~~~~~~~~~~~~~"<<endl;
    for (int i=0; i<4; i++){
        parts[i].start=*(int*)(buf + 0x08 +(i * 16));
        parts[i].size=*(int*)(buf + 0x0C + (i * 16));
        switch (buf[ 0x04 +(i * 16)]) {
            case 0x00 :
                parts[i].type="NOT-VALID";
                validParts--;
                break;
            case 0x06 :
                parts[i].type="FAT-16";
                break;
            case 0x07 :
                parts[i].type= "NTFS";
                break;
            case 0x0B:
```

```
                parts[i].type="FAT-32";
                break;
            default:
                parts[i].type="NOT-DECODED";
                break;
            }
        cout<<"Partition: "<<i<<"  Type: "<<parts[i].type<<"  Start:
"<<parts[i].start <<"  Size: "<<parts[i].size<<endl;
    }
    cout<<endl<<"Total number of valid partitions is: "<<validParts<<endl;
}

/**
 * @param: start - the entry of FAT16
 *
 * @brief: read the FAT-16 partition with the start offset,
 *         and print all of information
 *
 * @return: void
 */
void readFAT16(int start){
    char FAT16[64], delete_file[32], delete_file_content[16];
    char find_delete_file[2]; // read the first byte of every directory entry
to find the delete file
    int delete_file_size;
    int start_cluster;
    int CSA;
    file.seekg(start);
    file.read(FAT16,64);
    int num_sector = (int)(unsigned char)(FAT16[0x0D]);
    int size_FAT = (int)(FAT16[0x10]) * getOffset(FAT16, 0x16, 2);
    int size_reserved = getOffset(FAT16, 0x0E, 2);
    //cout<<"int size_reserved:    "<<size_reserved<<endl;
    int size_root_dir = getOffset(FAT16, 0x11, 2) * 32 / 512;
    int add_clu2= 63 + size_reserved + size_FAT +size_root_dir;

    cout<<"\n~~~~~~~~~~~~~~~~~~~~FAT-16
information~~~~~~~~~~~~~~~~~~~~"<<endl;

    cout<<"+++Basic info+++"<<endl;
    cout<<"number of sectors per cluster: "<<num_sector<<endl;
    cout<<"The size of the FAT area: "<<size_FAT<<endl;
    cout<<"The size of the Root directory: "<<size_root_dir<<endl;
    cout<<"The sector address of Cluster #2: "<<add_clu2<<endl;
```

```
    cout<<"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"<<end
l;
    cout<<"The information of first deleted file on the volume's root
directory:"<<endl;
    //root directory address is the data area start address, so it = (first sector
of volume) + (size of reserved area) + (size of FAT Area)
    int root_dir= (size_FAT + size_reserved + 63 )* 512;
    for (int i=root_dir; i<add_clu2*512; i+=0x20){
        file.seekg(i);
        file.read(find_delete_file,1);
        if ((int)(unsigned char)find_delete_file[0]==229){//the first byte of
deleted file is 0xE5
            file.seekg(i);
            file.read(delete_file,32);
            break;
        }
    }
    if ((int)(unsigned char)delete_file[0]!=229){
        cout<<"No deleted file on the volume's root directory!"<<endl;
        return;
    }
    cout<<"The name of the file: ";
    for (int i=0; i<11; i++){
        cout<<(delete_file[i]);
    }
    cout<<endl;
    delete_file_size = getOffset(delete_file, 28, 4);
    cout<<"the size of the file: "<<delete_file_size<<"
("<<hex<<delete_file_size<<") "<<endl;
    start_cluster = getOffset(delete_file, 26, 2);
    cout<<"the number of the first cluster: "<<dec<<start_cluster<<endl;
    CSA=add_clu2 + (start_cluster - 2) * num_sector;
    cout<<"The address of the cluster sector: "<<CSA<<endl;
    cout<<"The first 16 characters of the content of this file:"<<endl;
    file.seekg(CSA*512);
    file.read(delete_file_content, 16);
    for (int i=0; i<16; i++){
        cout<<(('!'<delete_file_content[i] &&
delete_file_content[i]<'~')?delete_file_content[i]:'.');
    }
    cout<<endl;

}
```

```
/**
 * @param: start - the entry of NTFS
 *
 * @brief: read the NTSF partition with the start offset,
 *              and print all of information
 *
 * @return: void
 */
void readNTFS(int start){
    char NTFS[84],MFT[1024];
    file.seekg(start);
    file.read(NTFS,84);  // Bytes 0x0B-0x53 are the BPB(25 bytes) and the
extended BPB(48 bytes).
    int byt_per_sector = getOffset(NTFS, 0x0B, 2);//0x0B  word   Bytes Per
Sector
    int sec_per_cluster=(int)(unsigned char)NTFS[0x0D]; //0x0D byte  Sectors
Per Cluster
    //0x30  long long(16 byte) Logical Cluster Number for the file $MFT
    long long addr;
    // do the left shift operation
    for(int i=7; i>=0; i--){
        if (i==7){
            addr=(int)(unsigned char)NTFS[0x30+i];
        }else{
            addr= addr<<8 | ((int)(unsigned char)NTFS[0x30+i]  & 0x000000ff) ;

        }
    }
    long sec_address = addr * sec_per_cluster;
    file.seekg(start+(sec_address * 512));
    file.read(MFT,1024);
    int fir_attri =  getOffset(MFT, 20, 2);
    int f_length = getOffset(MFT, fir_attri+4, 4);
    int f_type = getOffset(MFT, fir_attri, 4);
    int sec_attri = f_length + fir_attri;
    int s_length = getOffset(MFT, sec_attri+4, 4);
    int s_type = getOffset(MFT, sec_attri, 4);

    cout<<"\n~~~~~~~~~~~~~~~~~~~~~~NTFS
information~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cout<<"The bytes per sector for this NTFS volume: "<<byt_per_sector<<endl;
    cout<<"The sectors per cluster for this NTFS volume:
"<<sec_per_cluster<<endl;
```

```
    cout<<"The sector address for the $MFT file record: "<<sec_address<<endl;
    cout<<"The type and length of the first attribute in the $MFT record:
"<<endl;
    cout<<"type:  "<<f_type<<"\t length:  "<<f_length<<endl;
    cout<<"The type and length of the second attribute in the $MFT record:
"<<endl;
    cout<<"type:  "<<s_type<<"\t length:  "<<s_length<<endl;
}

int main() {

    char File_path[]="";
    string FilePath;
    bool NTFScount=false;
    bool FAT16count=false;
    char esc;
    cout <<"Please input the path of Image file: "<<endl;
    cin >> File_path;
    cout<<"-------------------------------------------------------------"<<end
l;
    readFile(File_path);
    readMBR();
    //count the number of FAT partition
    for (int i=0; i<4; i++){
        if (parts[i].type.compare("FAT-16")==0){
            readFAT16(parts[i].start * 512);
            FAT16count=true;
        }
    }
    //count the number of NTFS partition
    for (int i=0; i<4; i++){
            if (parts[i].type.compare("NTFS")==0){
                NTFScount=true;
                readNTFS(parts[i].start * 512);
            }
        }
    if (!NTFScount){

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<end
l;
        cout<<"There is no NTFS file system!"<<endl;
    }else if (!FAT16count){

    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<end
```

```
l;
        cout<<"There is no FAt-16 file system!"<<endl;
    }
    file.close();
    cout<<"\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<e
ndl;
    while(1){
    cout<<"please input q to exit this program!"<<endl;
    cin>>esc;
    if (esc == 'q'){
        break;}
    else{
        cout<<"error input!"<<endl;
    }
    }
    return 0;
}
```