**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**
**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**Khoa Công nghệ thông tin**

# AN TOÀN VÀ PHỤC HỒI DỮ LIỆU
# ĐỒ ÁN GIỮA KÌ

## Giảng viên: Thái Hùng Văn

**Sinh viên thực hiện:**



| MSSV | Họ tên |
|---|---|
| 20120083 | Nguyễn Trọng Hiếu |
| 20120144 | Lê Chí Nghĩa |

***Phân chia công việc:***

| ***Công việc*** | ***Người thực hiện*** |
|---|---|
| Lý thuyết | |
| Yêu cầu 1 | 20120144 |
| Yêu cầu 2 | 20120083 |
| Yêu cầu 3 | 20120144 |
| Yêu cầu 4 | 20120083 |
| Yêu cầu 5 | 20120144 |
| Yêu cầu 6 | 20120083 |
| Yêu cầu 7 | 20120144 |
| Thực hành | |
| Yêu cầu 1 | 20120083 |
| Yêu cầu 2 | 20120144 |
| Yêu cầu 3 | 20120083 |
| Yêu cầu 4 | 20120144 |
| Yêu cầu 5 | 20120083 |
| Yêu cầu 6 | 20120144 |
| Yêu cầu 7 | 20120083 |

***Tỉ lệ hoàn thành:***

| ***Công việc*** | ***Tỉ lệ hoàn thành*** |
|---|---|
| Lý thuyết | |
| Yêu cầu 1 | 100% |
| Yêu cầu 2 | 100% |
| Yêu cầu 3 | 100% |
| Yêu cầu 4 | 100% |
| Yêu cầu 5 | 100% |
| Yêu cầu 6 | 100% |
| Yêu cầu 7 | 100% |
| Thực hành | |
| Yêu cầu 1 | 100% |
| Yêu cầu 2 | 100% |
| Yêu cầu 3 | 100% |
| Yêu cầu 4 | 50% |
| Yêu cầu 5 | 100% |
| Yêu cầu 6 | 100% |
| Yêu cầu 7 | 80% |
| Tổng | 95% |

Mục lục

**Phần Lý thuyết**.....1

**Phần thực hành**.....4

**Tham khảo**.....11

## Phần Lý thuyết

**Xây dựng mô hình và thiết kế kiến trúc tổ chức cho một hệ thống tập tin được chứa trong file.**

*1/* Việc **bảo mật thông tin** (phòng tránh bị lộ nội dung 1 số tập tin quan trọng) được xem là thiết yếu nhất.

Bằng cách áp dụng biện pháp mã hóa file, folder, ta có thể mã hóa trực tiếp văn bản trước khi lưu xuống bộ nhớ. Tuy nhiên không khuyến khích làm việc trên, vì để mã hóa toàn bộ văn bản trước khi lưu xuống cần một lượng tính toán lớn, do đó quá trình ghi xuống và cũng như đọc nội dung lên rồi giải mã để có lại văn bản gốc sẽ tốn rất nhiều thời gian. Do đó, ta chỉ áp dụng việc mã hóa văn bản cho trường hợp cực kì cần thiết, thay vì vậy ta sẽ dùng biện pháp kiểm tra một đoạn code bất kì trước khi cho phép đọc file cũng như folder. Điều này sẽ ít tốn thời gian hơn và cũng thuận tiện hơn cho quá trình truy xuất dữ liệu.

2/ Việc **bảo vệ dữ liệu** (phòng tránh hư hỏng /mất mát nội dung các tập tin quan trọng) cũng rất cần thiết.

Ta sẽ cần đưa ra các biện pháp bảo vệ dữ liệu khi có các tác động không may từ người dùng hoặc lỗi trên hệ thống lưu trữ. Các đề xuất đưa ra có liên quan tới việc backup dữ liệu kết hợp với việc kiểm tra định kỳ để theo dõi tình trạng dữ liệu. Cụ thể hơn là tích hợp một số giao thức kiểm tra với quy trình so sánh thích hợp giữa bản backup với bản dữ liệu chính, hỗ trợ phân tích có lỗi và đưa ra hướng giải quyết bên trong giao thức.

3/ Việc **phục hồi dữ liệu** đã xóa theo hình thức **xóa bình thường** có tính khả thi cao *(thiết kế để hạn chế việc chúng bị chép đè)*, và tuyệt đối không thể phục hồi dữ liệu khi đã xóa theo hình thức **xóa không cho phục hồi**.

Cơ chế xóa trên FAT nói chung được thực hiện dưới hình thức ghi đè byte đầu tiên của entry chính/phụ ứng với file/folder, đồng thời thực hiện ghi đè giá trị rỗng trên các byte thể hiện cluster chứ data tương ứng (ứng với file/folder cần xóa). Có thể ứng dụng biện pháp trên cho 2 cơ chế xóa bình thường và xóa tạm thời được đề xuất.

Đối với việc ghi đè byte đầu tiên trong entry, ta có 2 trường hợp entry trống đó là (0x00: entry chưa từng được dùng và 0xE5: entry hiện không còn được dùng). Kế thừa dấu hiệu

nhận biết được nêu ra, ta có thể đưa ra các byte mới ứng với các trạng thái mong muốn như:

- Entry đã bị xóa nhưng còn có thể phục hồi: Đối với kiểu thiết kế này thì sẽ mất giá trị của byte đầu tiên (chứa giá trị liên quan đến tên file), về mặt nhận xét thì việc mất đi 1 Byte đầu so với việc tạo một entry mới thì đề xuất này có thể được đem vào áp dụng (mất một byte không đáng là bao). Tuy nhiên đối với việc thiết kế như vậy cũng chưa quá tối ưu trong quá trình tồn tại của file/folder do có thể người dùng xóa nhưng file/folder cũng không còn được dùng. Để khắc phục vấn đề đó thì ta có thể trích thêm một vài byte để đếm ngược tới ngày cuối cùng trước khi xóa hoàn toàn file. Cách thiết kế vẫn còn tùy thuộc vào mục đích cuối cùng của người yêu cầu.
- Entry đã bị xóa và không còn khả năng phục hồi
    - Xóa trong tình trạng không thể phục hồi mức một: Đề xuất dấu hiệu nhận biết cho Byte đầu tiên là file này đã được xóa hoàn toàn (có thể tận dụng dấu hiệu là giá trị 0x5E)
    - Xóa trong tình trạng không thể phục hồi mức hai: Kế thừa mức một, tiến hành xóa hoàn toàn các giá trị trong tất cả các entry của file/folder.
    - Xóa trong tình trạng không thể phục hồi mức ba: Kế thừa mức hai, đồng thời truy suất đến tất cả các dữ liệu được lưu trong các cluster liên quan đến file/folder, tiến hành ghi đè giá trị rác hoặc xóa hoàn toàn giá trị đó về 0x00.

4/ Việc **mã hóa dữ liệu** (với các dữ liệu cần bảo mật) phải đáp ứng các tiêu chí:
- Mã theo khóa là mật khẩu được thiết lập riêng biệt cho từng tập tin.
- Không lưu lại mật khẩu.
- Với cùng 1 nội dung và mật khẩu, mỗi lần mã hóa sẽ cho ra 1 bản mã có nội dung và chiều dài khác nhau.

Ta sẽ có có thể mã hóa dữ liệu như sau:

- Mã khóa thông tin theo mật khẩu: Có thể lưu trữ giá trị khóa tại metadata của file/folder với mỗi dữ liệu trong file/folder khi viết xuống hoặc đọc lên đều thông qua các thao tác với khóa (mã hóa, giải mã) bằng các giải thuật mã hóa như các phương pháp mã hóa đối xứng như Triple DES, Blowfish, Twofish,…

- Không lưu mật khẩu: có thể lưu mã hash của mật khẩu để thay thế cho mật khẩu (Do tính chất của hàm hash là hàm một chiều và có tính biến động cao). Thường thì mật khẩu sẽ được lưu tại metadata liên quan đến file/folder như Entry chính hoặc các vị trí có tầm quan trọng trong Disk/Volume. Khi đó nếu kẻ xấu có thể truy suất phần meta data đó thì học có thể khai thác được mật khẩu của file/folder, và từ đó kẻ gian có thể tiếp truy suất đến các file/folder khác nếu như cùng cài đặt chung mật khẩu.
- Có thể cài đặt với yêu cầu cùng nội dung và mật khẩu, mỗi lần mã hóa sẽ cho ra một bản mã có nội dung với chiều dài khác nhau bằng các giải thuật mã hóa tương ứng như Elgamal, các giải thuật block cipher mode như CBC, GCM, …

5/ Số tập tin cần tổ chức trong MyFS.DRS có thể đủ nhiều /đa dạng để **phải tổ chức hệ thống thư mục** phân cấp.

Để tổ chức hệ thống thư mục phân cấp, ta sẽ tham khảo mô hình file/folder của Hệ diều hành Windows, trong đó, một folder sẽ có thể chứa nhiều file hoặc nhiều folder nhỏ hơn theo phân cấp. Một file sẽ là một đơn vị nhỏ nhất, không thể chứa file khác hay folder mà sẽ chứa nội dung của chính nó.

6/ Nội dung dữ liệu bên trong file MyFS.DRS cần được truy xuất theo từng cụm 512 byte giống như sector trên các volume của HĐH hoặc các đĩa của máy tính. *(tức cần thiết kế & xây dựng các hàm ReadSector / WriteSector giống các hàm ReadSector / WriteSector của HĐH & máy tính).*

Nếu ta quản lí dữ liệu bên trong file MyFS.DRS theo đơn vị sector thì sẽ đảm bảo thỏa mãn tiêu chí truy xuất theo từng cụm 512 byte (1 sector = 512 byte).

File MyFS.DRS sẽ được tổ chức thành 4 phần:

- Boot Sector có kích thước 1 sector
- Vùng chứa bảng FAT có kích thước phụ thuộc vào giá trị nhập vào
- Vùng chứa bảng RDET có kích thước 128 sector
- Vùng Data sẽ nằm trong phần còn lại

Và trên cơ sở quản lí theo sector, ta cũng sẽ thiết kế nên các hàm ReadBlock và WriteBlock để đọc ghi trên phạm vi một sector trong một lần thao tác.

Ngoài ra, các entry trong bảng RDET sẽ được tổ chức theo một cấu trúc 32 byte như sau:

- Tên của file hoặc folder: byte 0 - 10
- Trạng thái: byte 11
    - Nếu là "A" – Archive thì đây là một file.
    - Nếu là "D" – Directory thì đây là một folder
- Không được sử dụng (Dùng cho các trường hợp lưu ngày giờ hoặc các thông tin đặc biệt sau này) : byte 12 - 23
- Vị trí cluster bắt đầu: byte 24 - 27
- Kích thước file: byte 28 – 31

7/ Bên trong file MyFS.DRS cần có 1 vùng dành riêng để backup các nội dung quan trọng.

Vùng Data sẽ được chia thành 2 vùng: Vùng lưu trữ dữ liệu thường và vùng lưu trữ dữ liệu backup. Trong đó, vùng lưu trữ dữ liệu thường sẽ chứa nội dung tất cả các file, folder dù có quan trọng hay không. Và khi một file hoặc folder có nội dung được xem là quan trọng thì một bản sao sẽ được tạo ra và lưu trữ vào bên trong vùng quan trọng. Nếu trong quá trình làm việc mà phát hiện một file hoặc folder bị lỗi trong vùng thường thì ta sẽ sử dụng tới dữ liệu trong vùng backup đó. Hoặc ngược lại, ta cũng cần các thuật toán để kiểm tra trong trường hợp dữ liệu trong vùng backup bị lỗi, từ đó tiến hành sao lưu dữ liệu backup từ vùng dữ liệu thường.

Tuy nhiên, việc cho kích thước của vùng backup càng lớn thì sẽ càng giảm khả năng lưu trữ của file MyFS.DRS, do đó, cần xem xét trên tình hình thực tế mà lựa chọn một tỉ lệ sao cho phù hợp nhất có thể.

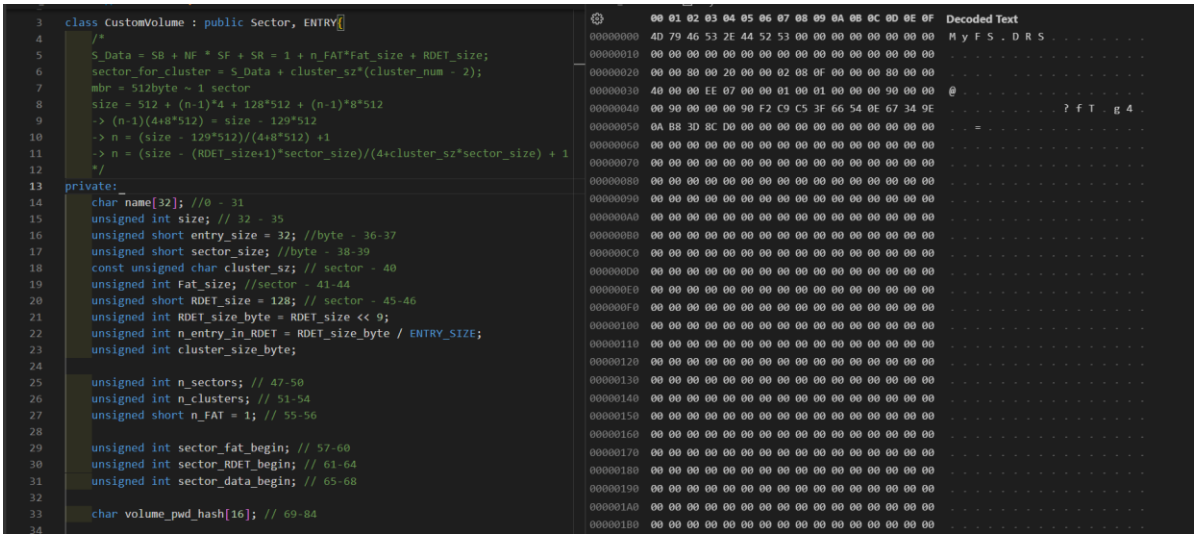
## Phần thực hành

Viết chương trình thực hiện các chức năng:

1/ Tạo / định dạng volume MyFS.DRS (với kích thước do người dùng tự nhập hoặc lựa chọn trong danh mục)

Volume có thết kế mô phỏng dựa trên mô hình hệ thống tập tin FAT của hệ điều hành. Tuy nhiên do giới hạn về mặt thời gian và tính khả thi khi thực hiện lab nên đã giới hạn lại một số thiết kế như:

- Chỉ tạo 1 bảng FAT (vẫn có thể tạo 2 bảng để đảm bảo các cơ chế kiểm tra hỏng cũng như cài đặt các giao thức kiểm tra lỗi và khôi phục lỗi – cần thời gian nghiên cứu và triển khai).
- Trên RDET mới chỉ hỗ trợ entry chính – cần thêm thời gian để thiết kế entry phụ, thông tin về ngày giờ cũng chưa được hỗ trợ.
- Boot sector được thiết kế cố định và bảng RDET được thiết kế kích thước cố định – 256 sectors. Kích thước bảng FAT được tính toán dựa trên kích thước của size truyền vào (công thức được thể hiện trong src code – volume.h).




*Hình 1: Thông số được thể hiện trong Master boot record và các giá trị thuộc tính tương ứng được khởi tạo trong class volume.h*

- Khởi tạo volume MyFS bằng class volume với kích thước volume được truyền vào cũng như giá trị password của volume, chưa tiến hành xử lý đối với password trống (có thể xảy ra lỗi – có thể fix nếu được cho thêm thời gian).
- Định dạng volume được định dạng bởi các thuộc tính private thể hiện trong class, các giá trị đó với kích thước (theo bytes) được đặt tại các vị trí (thể hiện dưới dạng comment ngay sau thuộc tính) ứng với vị trí trong sector đầu tiên được nêu trong code.

2/ Thiết lập /Đổi /Kiểm tra password truy xuất MyFS – nếu có thể thì hỗ trợ password đông /passkey

Password được lưu trữ dưới hình thức mã hash của password, đồng nghĩa với việc khi truy suất volume, password truyền vào sẽ được hash rồi đem giá trị hash đi so sánh với giá trị hash đã được lưu và phần metadata của volume. Nếu giá trị hash trùng với giá trị được lưu trong metadata thì có thể truy suất volume, nếu không thì không thể truy suất volume.

```
81  void CustomVolume::check_volpwd(const string& volname, const string& pwd){
82      if(this->openVolume(volname, pwd)){
83          cerr << "password of volume is correct" << endl;
84      }
85      else{
86          cerr << "password of volume is not correct" << endl;
87      }
88  }
89
90  void CustomVolume::update_volpwd(const string& volname) {
91      fstream file(volname, ios::binary | ios::in | ios::out);
92
93      if (!file.is_open()) {
94          cerr << "Error opening file: " << volname << endl;
95          return;
96      }
97
98      file.seekp(69, ios::beg);
99      file.write(&this->volume_pwd_hash[0], 16);
100     file.close();
101 }
102
103 void CustomVolume::change_volpwd(const string& volname, const string& old_
104     if(this->openVolume(volname, old_pwd)){
105         this->ComputeMD5(new_pwd, this->volume_pwd_hash);
106         this->update_volpwd(volname);
107         cerr << "change volume password successfully!" << endl;
108     }
109     else{
110         cerr << "old password not correct for changing!" << endl;
111     }
112 }
```

*Hình 2: Kiểm tra, thay đổi password - (việc thiết lập password được thực hiện khi khởi tạo volume).*

Ứng với cơ chế passkey, yêu cầu một kho chứa key nội bộ, cũng cần thiết kế bộ đăng nhập cho kho chưa key. Mô hình này có khuynh hướng đồng bộ hóa một account trên nhiều hệ thống khác nhau. Do đó chưa quá cần thiết để thiết kế một hệ thống passkey trên mô hình mô phỏng hệ thống tập tin trong phạm vi bài lab.

3/ Liệt kê danh sách các file trong MyFS

Liệt kê dưới hình thức kiểm tra các entry không trống trong bảng RDET, đánh dấu các entry không trống và thực hiện truy suất nội dung trên các entry đó để in thông tin tên file ra màn hình.

```cpp
void CustomVolume::listFile(const string& volname){
    this->readRDET(volname);
    vector<unsigned int> rdet_idx;
    char filename[32];

    for (unsigned int i = 0; i < n_entry_in_RDET; ++i){
        if(!this->is_empty(rdet[i]))
            rdet_idx.push_back(i);
    }

    for(auto idx = rdet_idx.begin(); idx != rdet_idx.end(); ++idx){
        memcpy(&filename, rdet[*idx], 32);
        cerr << "file: " << filename << endl;
        memset(&filename, 0x00, 32);
    }
}
```

*Hình 3: Kiểm tra các entry chính không trống và lấy tên các file/folder tương ứng.*

## 4/ Đặt /đổi password truy xuất cho 1 file trong MyFS

Thông tin của các tập tin được thể hiện trên FAT:

```
a.create_txt_file("test1.txt", 8*512+1, true);
a.create_txt_file("test2.txt", 8*512*2+1, true);

a.importFile("MyFS.DRS", "p@ssword", "vol_test1.txt", "test1.txt");
a.importFile("MyFS.DRS", "p@ssword", "vol_test2.txt", "test2.txt");

a.listFile("MyFS.DRS");
```

```
000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200  FF FF FF 0F FF FF FF 0F 03 00 00 00 FF FF FF 0F
00000210  05 00 00 00 06 00 00 00 FF FF FF 0F 00 00 00 00
00000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

*Hình 4: Thông tin trên FAT.*

Thông tin của các tập tin được thể hiện trên RDET:

```
00002000  76 6F 6C 5F 74 65 73 74 31 2E 00 20 00 00 00 00  v o l _ t e s t 1 . .
00002010  00 00 00 00 00 00 00 00 02 00 00 00 01 10 00 00  . . . . . . . . . . . .
00002020  76 6F 6C 5F 74 65 73 74 32 2E 00 20 00 00 00 00  v o l _ t e s t 2 . .
00002030  00 00 00 00 00 00 00 00 04 00 00 00 01 20 00 00  . . . . . . . . . . . .
```

*Hình 5: Thông tin trên các entry, nội dung cụ thể và các đọc thể hiện trong hình 6.*

Các cài đặp password cho file cũng tương tự như cho volume, tiến hành lưu trữ mã hash của password và tiến hành kiểm tra với các tương tác đến file. Chưa hoàn thành vì mặt thời gian tuy nhiên có thiết kế sẵn các hàm hash cho password và các byte tương ứng để lưu trữ giá trị hash (cần mở rộng phần entry chính để khả thi cho việc lưu trữ).

```
class ENTRY{
private:
    char name[11]; // 0-10
    char state; // 11

    char empty[12]; // not use  - 12-23


    unsigned int begin_cluster; // 24-27
    unsigned int file_size; //bytes - 28-31

    vector<pair<char, char>> state_table = {{ 'A', 0x20 }, {'D', 0x10}};
    /*
    A - Archive: file
    D - Directory: folder // not handle yet
    */
```

*Hình 6: Thiết kế của một entry mang tính mở rộng sang các entry chính/phụ khác khi cần thiết dựa trên 12 byte trống và 1 byte trạng thái còn có thể được tận dụng.*

```
00011FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00012000  53 65 62 32 35 7A 6E 7B 67 2C 4E 4C 58 44 58 7A  S e b 2 5 z n { g , N L X D X z
00012010  57 40 2B 49 41 36 5D 3A 53 55 7A 2C 46 3E 6E 21  W @ + I A 6 ] : S U z , F > n !
00012020  39 3C 66 66 28 5A 5B 3D 45 35 7E 50 75 3E 36 20  9 < f f ( Z [ = E 5 ~ P u > 6
00012030  2B 70 39 2F 73 4B 24 4B 47 2E 57 46 55 35 51 63  + p 9 / s K $ K G . W F U 5 Q c
00012040  51 4A 4C 5C 46 78 5F 36 31 62 66 65 2B 6A 6E 3B  Q J L \ F x _ 6 1 b f e + j n ;
00012050  62 44 2C 34 41 33 65 57 4E 4E 26 56 5F 3C 51 47  b D , 4 A 3 e W N N & V _ < Q G
00012060  22 75 62 4B 30 5C 28 3A 54 57 77 77 47 73 36 4A  " u b K 0 \ ( : T W w w G s 6 J
00012070  7E 67 3F 5D 50 40 78 72 63 55 49 79 6E 4B 75 75  ~ g ? ] P @ x r c U I y n K u u
00012080  64 4F 5B 72 79 75 66 62 37 2F 55 60 5F 67 35 64  d O [ r y u f b 7 / U ` _ g 5 d
00012090  21 31 63 3C 60 3B 45 25 74 78 47 75 78 60 75 4A  ! 1 c < ` ; E % t x G u x ` u J
000120A0  5C 6E 56 34 2F 34 74 7D 79 4F 58 59 2F 5D 67 27  \ n V 4 / 4 t } y O X Y / ] g '
000120B0  29 54 25 7A 3B 27 58 7B 62 36 51 25 33 64 78 39  ) T % z ; ' X { b 6 Q % 3 d x 9
000120C0  72 6F 28 21 20 76 3C 3E 55 26 33 6B 27 6D 7D 53  r o ( !  v < > U & 3 k ' m } S
000120D0  5E 4F 64 3E 20 67 63 27 2B 4B 3B 3B 20 3B 31 50  ^ O d >  g c ' + K ; ;  ; 1 P
000120E0  59 6E 60 37 5B 29 30 67 2E 6A 3D 72 69 62 50 61  Y n ` 7 [ ) 0 g . j = r i b P a
000120F0  34 72 7D 6A 55 68 46 43 42 59 45 34 63 61 40 71  4 r } j U h F C B Y E 4 c a @ q
00012100  71 31 7A 50 22 31 3A 23 4A 77 35 53 36 4D 2C 2A  q 1 z P " 1 : # J w 5 S 6 M , *
00012110  6E 5B 33 31 3D 57 34 4D 58 56 6B 76 44 79 39 5D  n [ 3 1 = W 4 M X V k v D y 9 ]
00012120  36 65 26 3A 6A 51 4E 57 40 49 41 75 21 3E 57 3B  6 e & : j Q N W @ I A u ! > W ;
00012130  2B 5E 4E 2C 22 28 4D 61 23 66 39 34 43 44 49 5C  + ^ N , " ( M a # f 9 4 C D I \
00012140  3D 5D 29 56 5C 56 52 2E 30 28 67 27 4E 2D 4C 51  = ] ) V \ V R . 0 ( g ' N - L Q
00012150  3C 5B 78 6E 44 59 35 33 44 2D 59 30 31 75 69 21  < [ x n D Y 5 3 D - Y 0 1 u i !
00012160  53 58 47 5E 48 50 6A 20 5B 77 3F 20 72 6C 4A 20  S X G ^ H P j  [ w ?  r l J
00012170  45 43 3D 6D 60 5D 63 67 51 5A 20 2C 22 77 40 4D  E C = m ` ] c g Q Z  , " w @ M
00012180  29 78 44 2C 41 30 32 49 5A 22 50 39 2A 3E 20 45  ) x D , A 0 2 I Z " P 9 * >  E
00012190  3E 76 50 3C 50 78 73 5C 64 3C 23 54 76 6A 4B 52  > v P < P x s \ d < # T v j K R
000121A0  4C 47 27 5C 64 28 7B 70 22 2C 42 52 63 7A 78 6F  L G ' \ d ( { p " , B R c z x o
000121B0  28 5A 37 7C 74 70 36 54 6D 3B 42 51 3E 34 3C 4D  ( Z 7 | t p 6 T m ; B Q > 4 < M
```

*Hình 7: Địa chỉ bắt đầu của vùng data là 12000 hexa.*

5/ Chép (Import) 1 file từ bên ngoài (từ hệ thống tập tin hiện hữu trên hệ điều hành đang sử dụng) vào MyFS

Cần xác định địa chỉ của file cần import và địa chỉ đích ở tập tin DRS để import vào. Vì phạm vi thực tế và các thiết kế mang tính khả thi trong phạm vi thời gian làm lab và tính khả thi của một mô hình an toàn cho hệ thống tập tin thì mã nguồn hiện tại chỉ cung cấp import file vào root của MyFS.DRS cũng như chưa hỗ trợ các thông tin liên quan đến folder như cài đặt entry chính cho folder hoặc import/export cho folder.

```
void CustomVolume::importFile(const string& volname, const string& volpwd,
                              const string& path_des_file, const string& path_src_file,
                              const bool is_folder, const string os){
    if(this->openVolume(volname, volpwd)){
        unsigned int file_sz = this->get_sizeof_file(path_src_file);
        if(file_sz == 0){
            return;
        }

        unsigned int n_cluster_need = ceil(static_cast<double>(file_sz) / this->cluster_size_byte);
        vector<unsigned int> clusters = this->find_n_empty_cluster(volname, n_cluster_need);
        if(clusters.size() == 0){
            cerr << "don't have enough cluster" << endl;
            return;
        }
        clusters.push_back(0x0FFFFFFF);

        char* entry = new char[32];
        char* cluster_data = new char[this->cluster_size_byte];

        ENTRY temp_entry(path_des_file, 'A', clusters[0], file_sz);
        temp_entry.get_entry(entry);

        this->writeEntry(volname, entry);

        for(unsigned int i = 0; i < clusters.size() - 1; i++){
            this->getFileData(path_src_file, file_sz, i, cluster_data);
            this->writeCluster(volname, clusters[i], cluster_data);
            this->set_cluster_value(volname, clusters[i], clusters[i+1]);
            file_sz -= this->cluster_size_byte;
            memset(cluster_data, 0x00, this->cluster_size_byte);
        }

        delete[] entry;
        delete[] cluster_data;
    }
    else{
        cerr << "password of volume is not correct" << endl;
    }
}
```

*Hình 8: Thực hiện lấy thông tin của file import và tính toán số lượng cluster cần thiết cũng như tìm vị trí các cluster - entry trống để thể hiện nội dung cho file/folder.*

6/ Chép (Outport) 1 file trong MyFS ra ngoài.

Yêu cầu địa chỉ của file nguồn (ở trong volume MyFS.DRS) và file đích (địa chỉ root của file main.exe). Chỉ tiến hành copy chứ không xóa file trong MyFS do cơ chế thiết lập (hoàn toàn có thể xóa file khi outport nếu có yêu cầu chỉnh sửa).

```
void CustomVolume::exportFile(const string& volname, const string& volpwd,
                              const string& path_des_file, const string& path_src_file,
                              const bool is_folder, const string os){
    if(this->openVolume(volname, volpwd)){
        unsigned int entry_index = 0;
        if(this->get_entry_with_filename(volname, path_src_file, entry_index)){
            unsigned int begin_cluster = 0;
            unsigned int file_size = 0;

            memcpy(&begin_cluster, rdet[entry_index] + 24, sizeof(begin_cluster));
            memcpy(&file_size, rdet[entry_index] + 28, sizeof(file_size));

            char* cluster_data = new char[this->cluster_size_byte];

            vector<unsigned int> clusters;
            unsigned int n_cluster_need = ceil(static_cast<double>(file_size) / this->cluster_size_byte);

            this->readFAT(volname);
            do{
                clusters.push_back(begin_cluster);
                begin_cluster = f[begin_cluster];
            } while(begin_cluster != this->my_pair[0].second);

            for(unsigned int i = 0; i < clusters.size(); i++){
                this->readCluster(volname, clusters[i], cluster_data);
                this->outFileData(path_des_file, file_size, i, cluster_data);

                file_size -= this->cluster_size_byte;
                memset(cluster_data, 0x00, this->cluster_size_byte);
            }
            delete[] cluster_data;
        }
        else{
            cerr << "Don't have file in volume" << endl;
        }
    }
    else{
        cerr << "password of volume is not correct" << endl;
    }
}
```

*Hình 9: Dựa vào entry chính, FAT data tương ứng với file/folder, thực hiện copy nội dung tới output file một cách tuần tự theo từng cluster.*

7/ Xóa 1 file trong MyFS (cho phép người dùng chọn dang không thể phục hồi hoặc có thể)

Chỉ thực hiện xóa file bình thường do giới hạn thời gian (các cơ chế về xóa file không thể phục hồi đã được nêu các thể hiện trong phần lý thuyết – hoàn toàn có thể thiết kế được), tiến hành ghi đè giá trị đầu của các entry tương ứng với file. Cơ chế xóa file hình thường được tiến hành bằng cách reset các cluster chứa nội dung file trong phần FAT và ghi đè giá trị đầu của entry về giá trị 0xE5.

```cpp
void CustomVolume::deleteFile(const string& volname, const string& volpwd,
                              const string& filename,
                              const bool is_folder, const string os){
    if(this->openVolume(volname, volpwd)){
        unsigned int entry_index = 0;
        if(this->get_entry_with_filename(volname, filename, entry_index)){
            unsigned int begin_cluster = 0;
            unsigned int file_size = 0;

            memcpy(&begin_cluster, rdet[entry_index] + 24, sizeof(begin_cluster));
            memcpy(&file_size, rdet[entry_index] + 28, sizeof(file_size));

            vector<unsigned int> clusters;
            unsigned int n_cluster_need = ceil(static_cast<double>(file_size) / this->cluster_size_byte);

            this->readFAT(volname);
            do{
                clusters.push_back(begin_cluster);
                begin_cluster = f[begin_cluster];
            } while(begin_cluster != this->my_pair[0].second);

            for(auto idx = clusters.begin(); idx != clusters.end(); ++idx){
                f[*idx] = 0x00000000;
            }
            this->update_fat(volname);

            rdet[entry_index][0] = 0xE5;
            this->update_rdet(volname);
        }
        else{
            cerr << "Don't have file in volume" << endl;
        }
    }
    else{
        cerr << "password of volume is not correct" << endl;
    }
}
```

*Hình 10: Lấy thông tin entry và FAT data liên quan đến file/folder, tiến hành xóa theo thiết lập.*

## Tham khảo

Nguồn tài liệu tham khảo của giảng viên hướng dẫn: link