**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO GIỮA KỲ**

**Học phần: Hệ điều hành**

**Lớp: CQ2019/4**

**Họ và tên các thành viên:**

1. **Bùi Lê Tuấn Anh – 19120163**
2. **Nguyễn Hồ Hoàng Duy - 19120064**
3. **Võ Phi Minh Hiếu - 19120075**

***Thành phố Hồ Chí Minh, tháng 11 năm 2021***

Nội dung

[BẢNG ĐÁNH GIÁ THÀNH VIÊN 2](#_Toc89677856)

[PHẦN 1. ĐỌC, TRUY XUẤT & PHỤC HỒI DỮ LIỆU 4](#_Toc89677857)

[PHẦN 2. THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG TẬP TIN 14](#_Toc89677858)

[TÀI LIỆU THAM KHẢO 25](#_Toc89677859)

# BẢNG ĐÁNH GIÁ THÀNH VIÊN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Công việc** | **Bùi Lê Tuấn Anh** | **Võ Phi Minh Hiếu** | **Nguyễn Hồ Hoàng Duy** |
| **Báo cáo** | | **1** | **1** | **1** |
| **1** | Viết báo cáo | x | x | x |
| **Bài 1** | | **0** | **1** | **2** |
| **2** | Viết chương trình 1B |  |  | x |
| **3** | Thực hiện câu 1A và 1C |  | x |  |
| **4** | Viết chương trình bat 1D và 1E |  |  | 50% |
| **Bài 2** | | **4** | **2** | **2** |
| **6** | Thiết kế hệ thống tập tin | x |  |  |
| **7** | Tạo/Định dạng Volume |  |  | x |
| **8** | Quản lý mật khẩu truy xuất | x |  |  |
| **9** | Liệt kê danh sách tập tin |  | x |  |
| **10** | Đặt mật khẩu tập tin | x |  |  |
| **11** | Chép tập tin từ bên ngoài vào |  | x |  |
| **12** | Chép tập tin từ Volume ra ngoai |  |  | x |
| **13** | Xóa 1 tập tin | x |  |  |
| **Tổng** | | **34%** | **33%** | **33%** |

***Đánh giá chung***: Dù gặp nhiều khó khăn khi liên lạc để thực hiện báo cáo nhưng cơ bản các thành viên hoàn thành nội dung đề ra.

# PHẦN 1. ĐỌC, TRUY XUẤT & PHỤC HỒI DỮ LIỆU

1. Bảng HexEdit của volume FAT32 đã format là:

**Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F**

**00000000** EB 58 90 4D 53 44 4F 53 35 2E 30 **00 02** **08** **0E 10** ëX.MSDOS5.0.....

**00000010** **02** **00 00** 00 00 F8 00 00 3F 00 FF 00 00 90 49 17 .....ø..?.ÿ...I.

**00000020 00 00 20 00** **F9 07 00 00** 00 00 00 00 02 00 00 00 .. .ù...........

**00000030** 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

**00000040** 80 00 29 10 B6 A6 7C 4E 4F 20 4E 41 4D 45 20 20 €.).¶¦|NO NAME

**00000050** 20 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4 FAT32 3ÉŽÑ¼ô

(Chỉ lấy 5 dòng đầu, do các thông tin cần thiết của câu A nằm trong vùng này.)

Xác định các thông số quan trọng:

* ***Số bytes của một sector*:** Theo lý thuyết bảng BootSector của FAT32, Số bytes của một sector xác định bởi 2 bytes tại offset **0000000B**. Theo bảng HexEdit trên, ta có 2 bytes đó là: **00 02**, mà máy tính hiện đang xét dùng hệ Little Endian nên sẽ dịch thành: **02 00**. Đổi qua dạng Decimal là: **512**. **Vậy một sector có 512 bytes.**
* ***Kích thước volume*:** Theo lý thuyết bảng BootSector của FAT32, Số sectors của một volume xác định bởi 4 bytes tại offset **00000020**. Theo bảng HexEdit trên, ta có 4 bytes đó là: **00 00 20 00**, mà máy tính hiện đang xét dùng hệ Little Endian nên sẽ dịch thành: **00 02 00 00**. Đổi qua dạng Decimal là: **2 097 152**. Vậy một volume có 2097152 sectors, với mỗi sector gồm 512 bytes. Vậy kích thước của Volume đang xét là: 2097152 \* 512 = 1073741824 bytes.

+ **Kích thước volume đang xét tính theo Megabytes là:**

1073741824/1000000 = **1073.741824 MB**. (Do 1 Megabyte = 1 000 000 bytes)

+ **Kích thước volume đang xét tính theo Mebibytes là:**

1073741824/(1024^2)= **1024 MiB.** (Do 1 Mebibytes = 1024^2 bytes )

* ***Số sector trước FAT*:** Theo lý thuyết bảng BootSector của FAT32, Số sector trước bảng FAT (hay số sector thuộc vùng BootSector) xác định bởi 2 bytes tại offset **0000000E**. Theo bảng HexEdit trên, ta có 2 bytes đó là: **0E 10**,mà máy tính hiện đang xét dùng hệ Little Endian nên sẽ dịch thành: **10** **0E**. Đổi qua dạng Decimal là: **4110. Vậy số sector trước bảng FAT là 4110 sectors.**
* ***Số sector của một bảng FAT và số bảng FAT*:** Theo lý thuyết bảng BootSector của FAT32, Số sector của một bảng FAT xác định bởi 4 bytes tại offset **00000024**, còn số bảng FAT xác định bởi 1 byte tại offset **00000010**. Theo bảng HexEdit trên, ta lần lượt có:

+ 4 bytes tại offset **00000024** là: **F9 07 00 00**,mà máy tính hiện đang xét dùng hệ Little Endian nên sẽ dịch thành **00 00 07 F9**, chuyển qua Decimal là: **2041**. **Vậy số sector của một bảng FAT là: 2041 sectors.**

+ 1 byte tại offset **00000010** là: **02** thì vẫn là **02** khi chuyển qua Little Endian và chuyển qua Decima thì là **2**. **Vậy số bảng FAT là 2.**

* ***Số entry hiện tại của RDET*:** Theo lý thuyết bảng BootSector của FAT32, số entry của RDET xác định bởi 2 bytes tại offset **00000011**. Theo bảng HexEdit trên, ta có 2 bytes đó là: **00 00**, chuyển qua Decimal thì là: **0**. **Vậy số entry của RDET là 0.**
* ***Số sector của một cluster*:** Theo lý thuyết bảng BootSector của FAT32, số sector của một cluster xác định bởi 1 bytes tại offset **0000000D**. Theo bảng HexEdit trên, ta có 1 bytes đó là: **08**, chuyển qua Decimal là: **8**. **Vậy một cluster sẽ có 8 sectors.**
* **Số cluster của volume:** Ở trên ta đã biết được volume đang xét có 2097152 sectors và một cluster sẽ có 8 sectors. **Vậy số cluster của volume này là: 2097152/8 = 262144 cluster.**
* **Công thức tính vị trí của cluster theo sector:** Theo slide lý thuyết Tổ chức hệ thống tập tin FAT, ta có công thức tương quan vị trí của cluster thứ k trên vùng dữ liệu và số sector là:
  + **i = SB + SF \* NF + [SRDET] + (k-2) \* SC**

Trong đó:

+ i : là số thứ tự của sector tương ứng với vị trí bắt đầu của cluster.

+ SB : là số sector vùng BootSector

+ SF : là số sector một bảng FAT

+ NF : là số bảng FAT

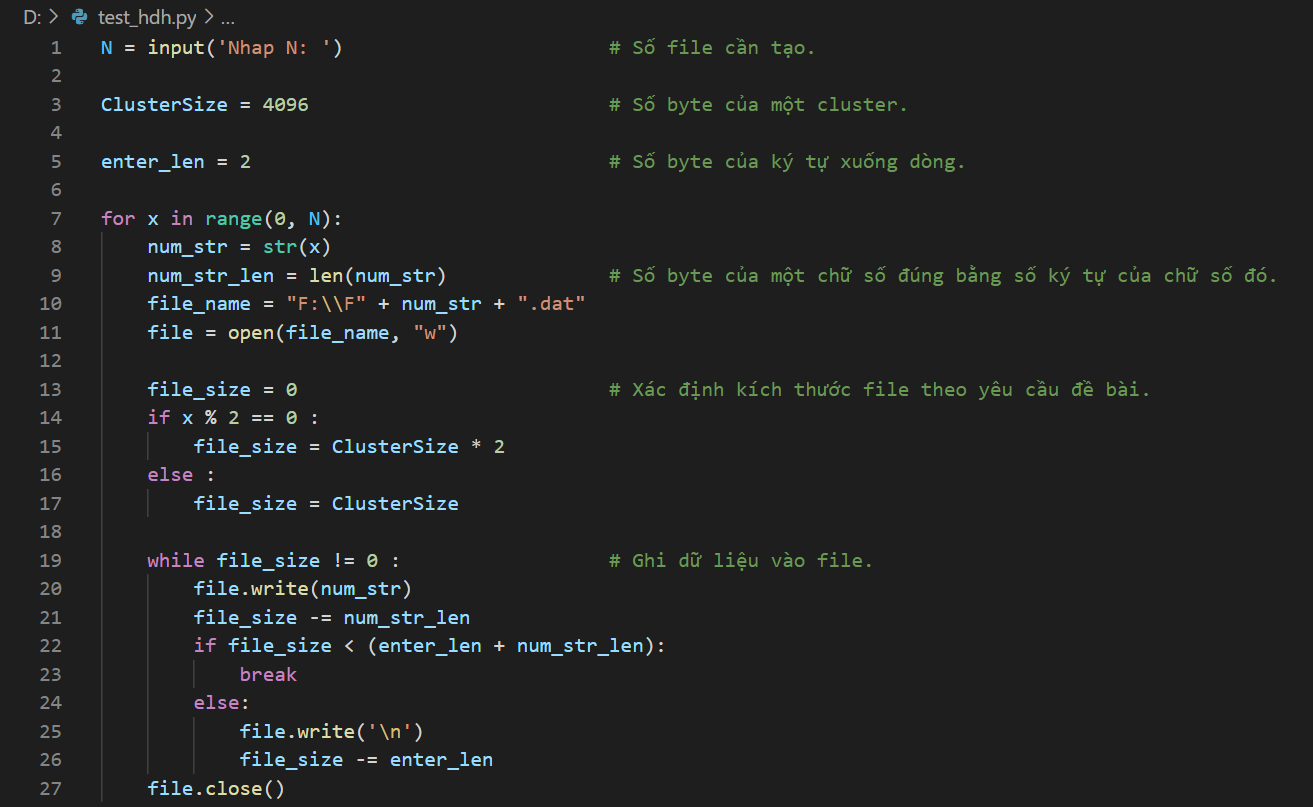
+ SRDET : là số sector của bảng RDET. Trong FAT32 thì không có thông số này, do bảng thư mục gốc RDET nằm trên vùng dữ liệu và kích thước có thể thể thay đổi.

+ SC : là số sector của một cluster.

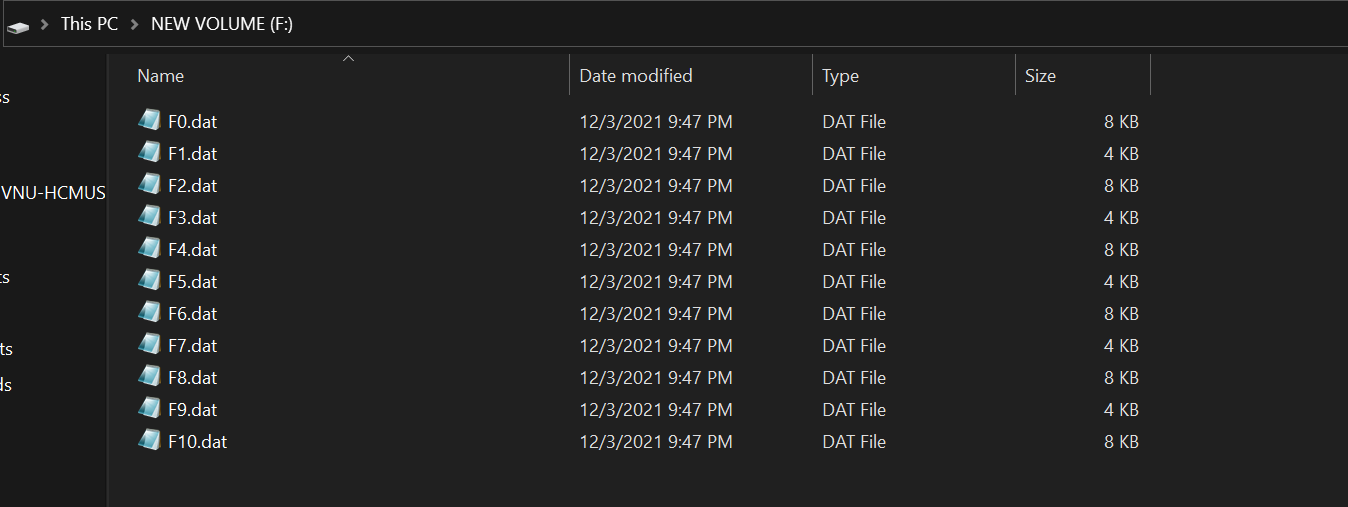
Ví dụ: Theo các bảng HexEdit trên, cluster thứ 2 sẽ bắt đầu tại sector thứ:

+ i = 4110 + 2041 \* 2 + 0 + (2-2) \* 8 = 8192.

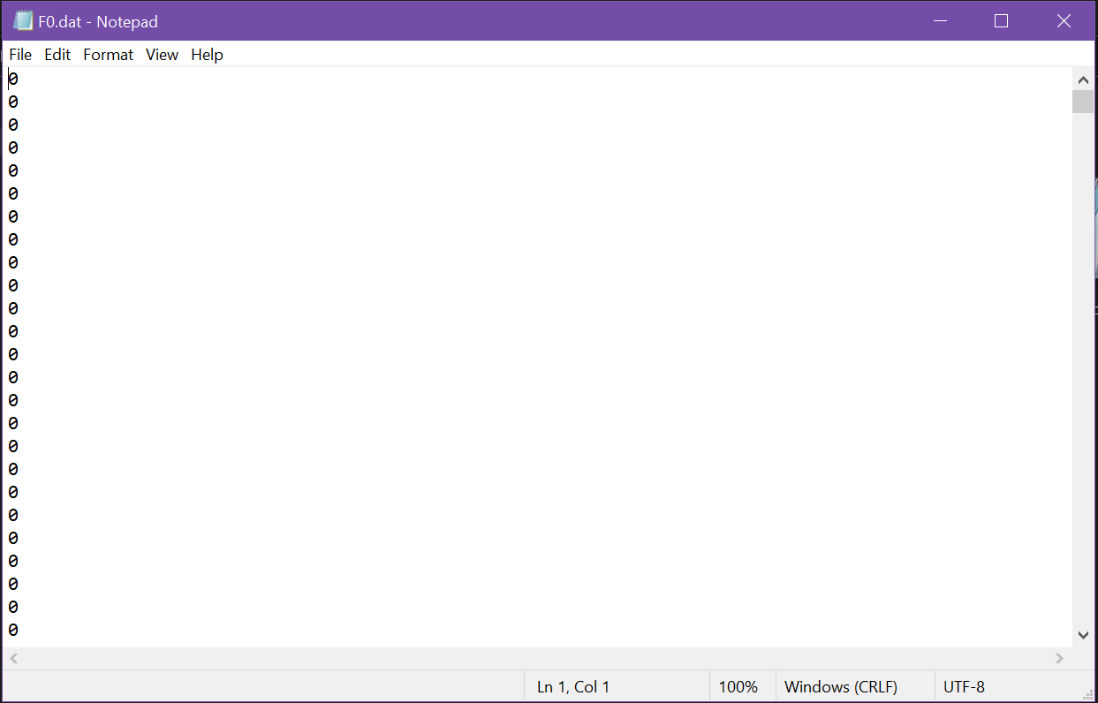
1. Chương trình đưa vào (thư mục gốc) N tập tin F0.Dat, F1.Dat,… F<N>.Dat với nội dung của F<K>.Dat là các dòng văn bản mà mỗi dòng đều là giá trị K và chiếm (2 - K%2) cluster:



Kết quả với N = 11:



Bên trong File F0.dat:



Giải thích:

* **Về Clustersize:** Ở câu A, nhóm đã xác định được số sector của một cluster là 8 và kích thước của một sector là 512 bytes. Nên tính được số bytes của 1 cluster là 512 \* 8 = 4096 bytes.
* **Về nội dung code:** Đầu tiên nhóm đặt một biến file\_size là kích thước của file bằng 1 clustersize hoặc 2 clustersize theo yêu cầu đề. Sau đó bắt đầu ghi vào file các ký tự chữ số và dấu xuống dòng cũng theo yêu cầu đề, rồi giảm giá trị của biến file\_size xuống tương ứng với số byte đã ghi vào file. Đến khi biến file\_size bằng 0 hoặc nhỏ hơn kích thước của chữ số cần ghi vào thì dừng, điều này đảm bảo được kích thước file sẽ luôn là 1 cluster hoặc 2 cluster.

1. Dự đoán (có lý giải) số cluster của RDET khi:

* N là 11:
* Theo lý thuyết bảng BootSector của FAT32, ta có cluster bắt đầu của RDET được xác định bởi 4 bytes ở offset **0000002C,** theo bảng HexEdit ở câu A thì 4 bytes đó là: 02 00 00 00, dịch theo Little Endian sẽ là 00 00 00 02. Chuyển qua Decimal sẽ là: 2. Vậy bảng RDET sẽ bắt đầu ở cluster 2. Lại tiếp tục tính sector bắt đầu của cluster 2 theo thông tin ổ đĩa như câu A, ta có: i = 4110 + 2041 \* 2 + 0 + (2-2) \* 8 = 8192. Vậy bảng RDET sẽ bắt đầu tại sector 8192. Nhảy tới sector này trong HexEdit, ta được bảng sau:

**Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F**

00400000 4E 45 57 20 56 4F 4C 55 4D 45 20 08 00 00 00 00 NEW VOLUME .....

00400010 00 00 00 00 00 00 D0 66 84 53 00 00 00 00 00 00 ......Ðf„S......

00400020 42 20 00 49 00 6E 00 66 00 6F 00 0F 00 72 72 00 B .I.n.f.o...rr.

00400030 6D 00 61 00 74 00 69 00 6F 00 00 00 6E 00 00 00 m.a.t.i.o...n...

00400040 01 53 00 79 00 73 00 74 00 65 00 0F 00 72 6D 00 .S.y.s.t.e...rm.

00400050 20 00 56 00 6F 00 6C 00 75 00 00 00 6D 00 65 00 .V.o.l.u...m.e.

00400060 53 59 53 54 45 4D 7E 31 20 20 20 16 00 5A CF 66 SYSTEM~1 ..ZÏf

00400070 84 53 84 53 00 00 D0 66 84 53 03 00 00 00 00 00 „S„S..Ðf„S......

00400080 46 30 20 20 20 20 20 20 44 41 54 20 10 87 EB 66 F0 DAT .‡ëf

00400090 84 53 84 53 00 00 EC 66 84 53 05 00 FF 1F 00 00 „S„S..ìf„S..ÿ...

004000A0 46 31 20 20 20 20 20 20 44 41 54 20 10 88 EB 66 F1 DAT .ˆëf

004000B0 84 53 84 53 00 00 EC 66 84 53 07 00 00 10 00 00 „S„S..ìf„S......

004000C0 46 32 20 20 20 20 20 20 44 41 54 20 10 88 EB 66 F2 DAT .ˆëf

004000D0 84 53 84 53 00 00 EC 66 84 53 08 00 FF 1F 00 00 „S„S..ìf„S..ÿ...

004000E0 46 33 20 20 20 20 20 20 44 41 54 20 10 88 EB 66 F3 DAT .ˆëf

004000F0 84 53 84 53 00 00 EC 66 84 53 0A 00 00 10 00 00 „S„S..ìf„S......

00400100 46 34 20 20 20 20 20 20 44 41 54 20 10 88 EB 66 F4 DAT .ˆëf

00400110 84 53 84 53 00 00 EC 66 84 53 0B 00 FF 1F 00 00 „S„S..ìf„S..ÿ...

00400120 46 35 20 20 20 20 20 20 44 41 54 20 10 89 EB 66 F5 DAT .‰ëf

00400130 84 53 84 53 00 00 EC 66 84 53 0D 00 00 10 00 00 „S„S..ìf„S......

00400140 46 36 20 20 20 20 20 20 44 41 54 20 10 89 EB 66 F6 DAT .‰ëf

00400150 84 53 84 53 00 00 EC 66 84 53 0E 00 FF 1F 00 00 „S„S..ìf„S..ÿ...

00400160 46 37 20 20 20 20 20 20 44 41 54 20 10 89 EB 66 F7 DAT .‰ëf

00400170 84 53 84 53 00 00 EC 66 84 53 10 00 00 10 00 00 „S„S..ìf„S......

00400180 46 38 20 20 20 20 20 20 44 41 54 20 10 89 EB 66 F8 DAT .‰ëf

00400190 84 53 84 53 00 00 EC 66 84 53 11 00 FF 1F 00 00 „S„S..ìf„S..ÿ...

004001A0 46 39 20 20 20 20 20 20 44 41 54 20 10 89 EB 66 F9 DAT .‰ëf

004001B0 84 53 84 53 00 00 EC 66 84 53 13 00 00 10 00 00 „S„S..ìf„S......

004001C0 46 31 30 20 20 20 20 20 44 41 54 20 10 8A EB 66 F10 DAT .Šëf

004001D0 84 53 84 53 00 00 EC 66 84 53 14 00 FE 1F 00 00 „S„S..ìf„S..þ...

004001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

004001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

* Theo lý thuyết, bảng RDET của FAT32 không có giới hạn và nằm trên vùng dữ liệu, cho nên độ lớn của nó phụ thuộc số mục mà nó lưu trữ, với mỗi mục đều có độ lớn là 32 bytes. Nhóm nhảy tới bảng RDET và biết được ngoài các mục lưu trữ thì bảng còn có 8 dòng = 128 bytes ở đầu.
* Nhóm dự đoán kích thước của RDET khi N = 11 sẽ là 128 + 11\*32 = 480 bytes < 4096 bytes. **Vậy kích thước của RDET sẽ gói gọn trong 1 cluster.** Và cũng đã kiểm chứng với bảng trên.
* N là 2021:
* Tương tự N = 11, nhóm dự đoán kích thước của RDET khi N = 21 sẽ là: 128 + 2021 \*32 = 64800 bytes. Với mỗi cluster = 4096 bytes, kích thước của RDET là: 64800/4096 = 15.8203125 (xấp xỉ 16). **Vậy nhóm dự đoán kích thước của RDET khi N = 21 sẽ là 16 cluster.**

D. Xóa toàn bộ các file F<K>.Dat với K chẵn bằng cách tạo và chạy một file .BAT với vòng lặp FOR bên trong (nếu không thể làm được thì xóa bằng tay qua công cụ File Explorer của HĐH). Sau đó cứu F0.dat bằng cách dùng *HexEdit* (*diễn giải chi tiết các nội dung được điều chỉnh trên sector /các nội dung sector có sử dụng cần đưa vào trong báo cáo)*

* Xóa file F<K>.Dat theo yêu cầu:
* *Nội dung của file bat:*

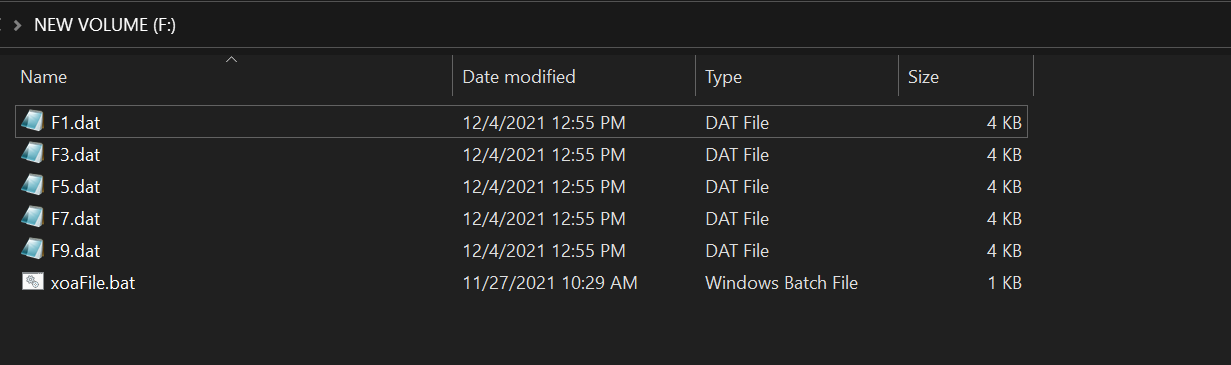
@echo off

FOR /L %%i IN (0,2,%1) DO (del F%%i.dat)

* + *Cú pháp khởi chạy file trên cmd:* F:\>xoaFile.bat N

+ Với N là số file trong volume.

* *Kết quả sau khi chạy với N = 11:*



* **Cứu F0.dat bằng HexEdit:**

*1/ Trong bảng RDET:*

* Thông số F0.dat trong bảng RDET trước khi chạy:

**Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F**

**...**

**00400080 46 30 20 20 20 20 20 20 44 41 54 20 10 87 EB 66 F0 DAT .‡ëf**

**00400090 84 53 84 53 00 00 EC 66 84 53 05 00 FF 1F 00 00 „S„S..ìf„S..ÿ...**

**…**

* Thông số F0.dat trong bảng RDET sau khi chạy:

**Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F**

**...**

**00400080 E5 30 20 20 20 20 20 20 44 41 54 20 10 87 EB 66 å0 DAT .‡ëf**

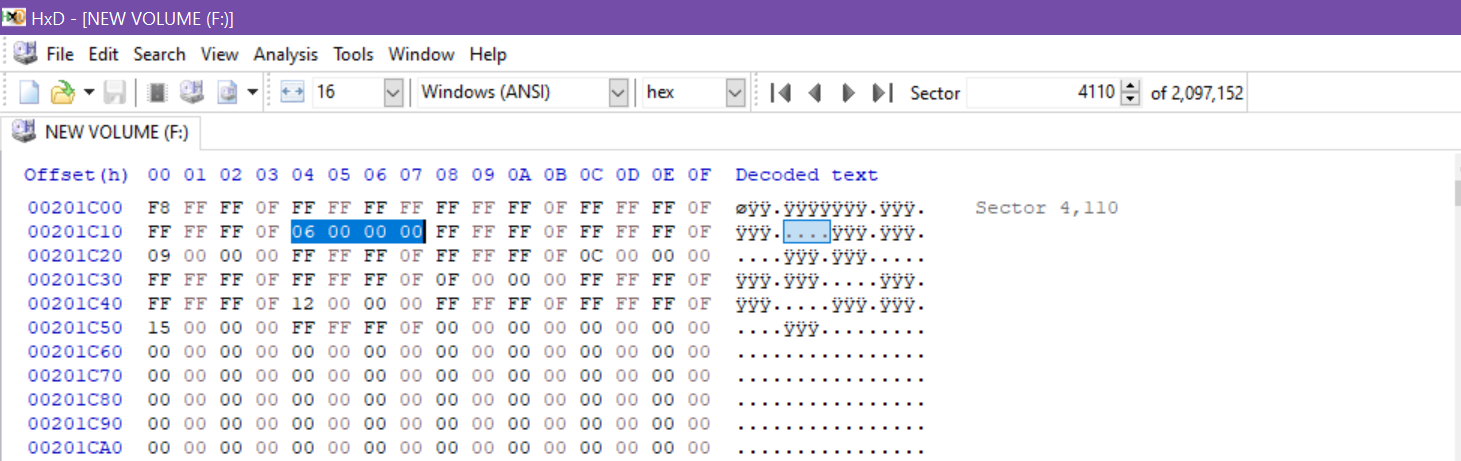
**00400090**  **84 53 84 53 00 00 EC 66 84 53 05 00 FF 1F 00 00**  **„S„S..ìf„S..ÿ...**

**...**

* Theo lý thuyết bảng entry chính của RDET, mỗi một file hoặc thư mục sẽ được lưu thông tin cơ bản trong bảng RDET dưới dạng 1 mục 32 bytes.
* So sánh thông số của mục F0.dat trước và sau khi chạy file xoaFile.bat để xóa, nhóm nhận thấy các dữ liệu đa phần đều giữ nguyên, trừ byte đầu tiên của mục. Theo lý thuyết, byte đầu tiên của mục sẽ lưu trạng thái hiện tại của mục đó trong bảng RDET. Và sau khi mục đã xóa, thì byte đầu tiên sẽ chuyển thành E5 như trên, để thông báo rằng có thể lưu đè lên vùng dữ liệu đó.
* **Nhóm xác định cần sửa byte đầu tiên trong mục F0.dat của bảng RDET để khôi phục trạng thái của mục thành có thể đọc.**

*2/ Trong bảng FAT:*

* Mỗi phần tử trong bảng FAT của FAT32 sẽ gồm: 32 bits = 4 bytes. Mỗi phần tử này sẽ đại diện cho 1 cluster tương ứng, với phần tử bắt đầu là 0.
* Theo lý thuyết thông số entry chính của bảng RDET, 2 bytes từ offset 1A (nghĩa là offset A ở hàng thứ 2) sẽ xác định cluster bắt đầu của phần nội dung tập tin. Xét trong mục F0.dat thì offset tương ứng là 0040009A và 2 bytes tương ứng là: 05 00 Dịch qua Little Endian sẽ là 00 05, chuyển về Decimal là: 5. Vậy phần nội dung file F0.dat bắt đầu tại cluster 5. Và bao gồm 2 cluster là 5 và 6. Do file F0.dat có độ lớn là 2 cluster.
* Nhóm nhảy tới vị trí của bảng FAT ở sector 4110 thì thấy thông số ở 2 phần tử 5 và 6 trong bảng (tương ứng cluster 5, cluster 6 là vùng mà dữ liệu F0.dat chiếm) đều bị đổi thành 0. Vậy nghĩa là muốn khôi phục lại F0 thì phải khôi phục cả giá trị của phần tử 5 và 6 trên bảng FAT về như cũ.
* Xét bảng FAT trước khi xóa, đối với các file chiếm 2 cluster ta thấy:



+ 4 bytes phần tử đầu sẽ là một giá trị bằng số thứ tự của phần tử cộng thêm 1 đơn vị. Ví dụ: ở phần tử thứ 5 trong bảng FAT là: **06 00 00 00.** Phần tử thứ 8 trong bảng FAT là **09 00 00 00 ...**

+ 4 bytes phần tử sau sẽ là **FF FF FF 0F.** Vì theo bảng giá trị các phần tử FAT thì giá trị này nghĩa là cluster cuối của tập tin.

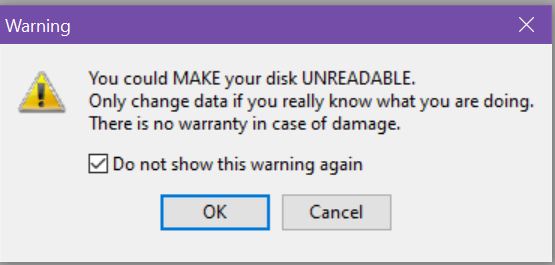
* **Vậy muốn khôi phục F0.dat thì phải sửa 2 phần tử 5 và 6 trong bảng FAT về giá trị ban đầu.**

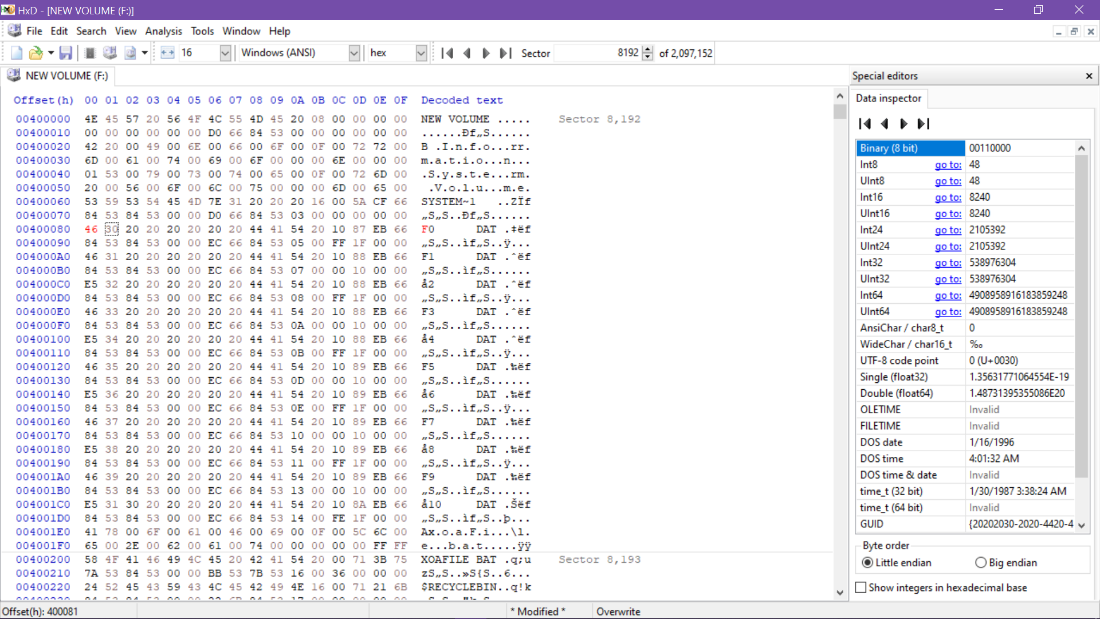
*3/ Về nội dung file:*

* Ở trên, nhóm đã xác định được cluster bắt đầu của phần nội dung file F0.dat. Dùng công thức liên quan giữa cluster và sector ở câu A, nhóm nhảy đến sector: 8216 trong bảng HexEdit. Nhóm nhận thấy dữ liệu của file còn nguyên vẹn. **Vậy chỉ cần sửa đổi 2 phần là RDET và FAT là được.**

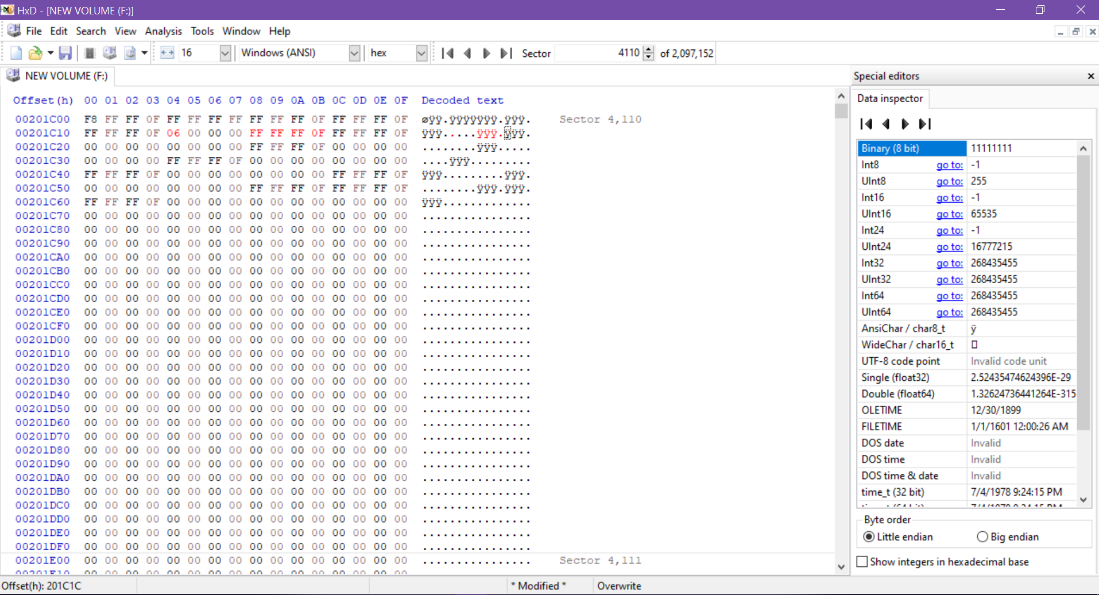
Graphical user interface, text, application, email

Description automatically generated*4/ Quy trình sửa đổi:*

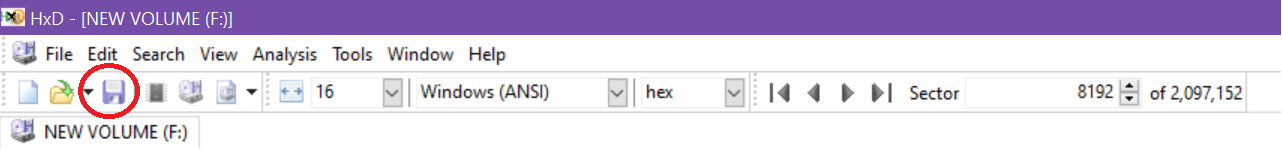
* Để chỉnh sửa ổ đĩa, ta chọn Tool > Option, bỏ chọn “Open disks as readonly by default”. Rồi nhấn OK.
* Sau đó thoát HxD và mở lại theo dạng Run as Adminitrator, chọn volume đang định sửa, hộp thoại cảnh báo hiện lên, nhấn OK.
* Tìm đến bảng RDET ở sector 8192, sau đó chỉnh sửa ký tự đầu tiên của mục F0.dat trở lại thành ‘**F**’.



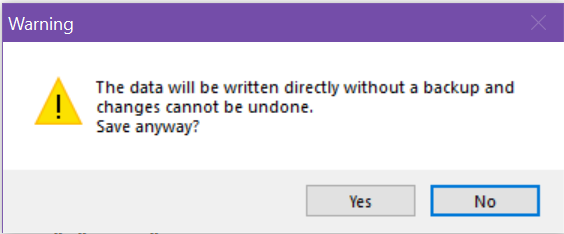
* Tìm đến bảng FAT ở sector 4110, sau đó chỉnh sửa phần tử 5 và 6 của bảng thành: ‘**06 00 00 00 FF FF FF 0F**’



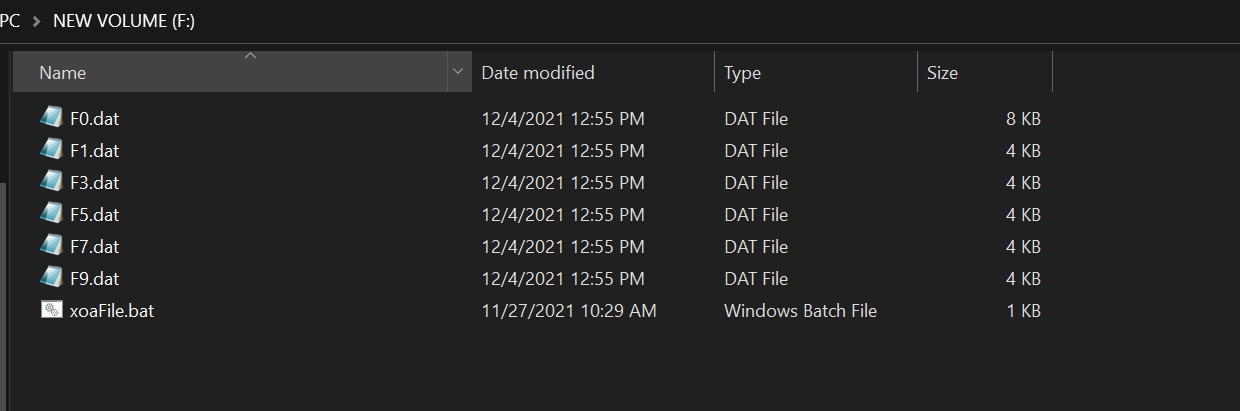
* Nhấn chọn biểu tượng ‘Save’:



* Hộp thoại cảnh báo hiện lên, nhấn chọn Yes:



* Vậy là đã khôi phục F0.dat thành công, kết quả:



# PHẦN 2. THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG TẬP TIN

1. Để xây dựng một hệ thống tập tin đáp ứng yêu cầu như file MyFS.dat, dựa trên hướng dẫn của phần [tài liệu tham khảo](#_TÀI_LIỆU_THAM) (mục 1) và sự hiểu biết của các thành viên, nhóm quyết định triển khai theo mô hình như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **BOOT** | **INFO** | **ENTRY** | **DATA** |
| **Siêu dữ liệu hệ thống** | | | **Phần dữ liệu** |

Sơ lược về các vùng như sau **(1 sector tương ứng với 512 bytes)**:

1. **Vùng BOOT**: Vùng này có kích thước 1 sector **(sector 0)**. Vùng này chứa các thông tin liên quan đến file MyFS.dat. Vùng này khá giống với một Boot Sector trên Hệ điều hành thực tế, chuyên dùng để đọc dữ liệu của MyFS.dat. Vùng này đóng vai trò vô cùng quan trọng, bởi nếu không có vùng này, hệ điều hành sẽ không thể nhận diện được file MyFS.dat
2. **Vùng INFO:** Vùng này có kích thước 1 sector **(sector 1)**. Vùng này có nhiệm vụ sẽ lưu toàn bộ thông tin của hệ thống tập tin trong file MyFS.dat khi chạy trên phần cứng.
3. **Vùng ENTRY:** Đây là phân vùng có vai trò khá quyết định đến hệ thống tập tin, bởi lẽ đây chính là nơi để quản lý các tập tin bên trong hệ thống. Kích thước dự kiến của vùng này sẽ kéo từ **sector 2 đến sector 1024**. Bên trong mỗi sector sẽ chia thành **16 phần bằng nhau**, mỗi phần có kích thước **32 bytes**. Bên trong mỗi phần sẽ lưu các thông số sau:

|  |  |  |
| --- | --- | --- |
| **Cộng** | **Độ lớn (bytes)** | **Nội dung** |
| 0 | 3 | Thời gian chỉnh sửa lần cuối |
| 3 | 2 | Ngày chỉnh sửa lần cuối |
| 5 | 4 | Kích thước dữ liệu của file/thư mục |
| 9 | 1 | Kiểu dữ liệu |
| 10 | 1 | Độ dài mật khẩu |
| 11 | 1 | Mật khẩu của file/thư mục |
| 12 | 20 | Danh sách các sector của file/thư mục |

Một số điều cần chú ý về các thuộc tính này như sau:

* **Thời gian chỉnh sửa lần cuối**: Xét một khung giờ nhất định theo định dạng HH:MM:SS, ta thấy:
  + HH: Giờ, từ 00 – 23. Tối thiểu là 5 bit biểu diễn
  + MM: Phút, từ 00 – 59. Tối thiểu là 6 bit biểu diễn
  + SS: Giây, từ 00 – 60 (áp dụng theo quy chuẩn UTC, giây nhuận theo quốc tế sẽ được cộng vào một số thời điểm, giúp gia tăng độ chính xác). Tối thiểu là 6 bit.
  + MS: Mili giây, từ 00-99. Tối thiểu là 7 bit biểu diễn.
* **Ngày chỉnh sửa lần cuối**: Xét một định dạng ngày cụ thể là DD-MM-YYYY, ta thấy:
  + DD: Ngày, từ 01 – 31. Tối thiểu là 5 bit (thấp)
  + MM: Tháng, từ 01 – 12. Tối thiểu là 4 bit (giữa)
  + YYYY: Năm. Thường xác định từ năm 1980. Dùng 7 bit cao (Như vậy thời gian sẽ đến năm 2107, sau này, nếu như tiếp tục sử dụng hệ thống tập tin này, có thể điều chỉnh thông số của vùng này ở những phiên bản mới hơn).
* **Kích thước dữ liệu của File/Thư mục**: Dùng 4 byte biểu diễn. Như vậy kích thước mỗi file sẽ ở khoảng 232 bytes (khoảng 4 tỷ bytes, nếu tính theo quy ước sẽ ở **khoảng 4GB**). Kích thước này phù hợp với tiêu chí số 5, không phải tất cả mọi file đều lớn hơn 4GB.
* **Độ dài mật khẩu**: Lưu độ dài, phục vụ truy xuất Mật khẩu file/Thư mục
* **Kiểu dữ liệu:** Thuộc tính này sử dụng 1 byte để kiểm tra loại dữ liệu tương ứng. Nhóm khám phá ra, đó là việc một thư mục cũng là một file. Do đó nhóm sẽ tạo ra 1 kiểu dữ liệu mang tên: ‘Thư mục’ để tiến hành lưu trữ. Điều đáng chú ý hơn sẽ nằm ở vùng **DATA**.
* **Mật khẩu file/thư mục:** Tiêu chí được đặt lên hàng đầu là bảo mật. Do đó nhóm chủ trương áp dụng công nghệ mã hóa hiện đại nhất là **SHA-256**, tiến hành băm mật khẩu nhiều lần trước khi lưu lại. Độ dài m, truy xuất từ thuộc tính “Độ dài mật khẩu”. Như vậy, chiều dài tối đa là (28 – 1), tức là 255 bit (khoảng **32 bytes**, tương đương một chuỗi số nguyên độ dài 8).
* **Danh sách các sector:** Đây là một thuộc tính rất đặc biệt. Thuộc tính này sẽ lần lượt lưu các giá trị là vị trí các sector, theo thứ tự của dữ liệu.
  + Hiển nhiên, dữ liệu trên mỗi file không lưu liên tiếp nhau, thay vào đó là lưu chia nhỏ thành nhiều sector. 20 bytes = 160 bits, vừa đủ để lưu giá trị khoảng 10 con số khác nhau liên tiếp.
  + Tại thuộc tính này, ta có thể sử dụng cơ chế **địa chỉ gián tiếp** để lưu các số đầu là các sector, còn số cuối là sector dẫn đến sector khác.
  + Tại mỗi sector ta có thể lưu dữ liệu của file, hoặc có thể lưu danh sách các sector, tức là sử dụng cơ chế **địa chỉ gián tiếp cấp 2** để lưu (lúc này ta có thể lưu tới **170-256** con số khác nhau). Nếu tính một cách chuẩn xác, ta có thể lưu một file có kích thước cỡ khoảng **20-30MB** hoặc nhiều hơn, tùy thuộc vào cấp độ truy xuất. Điều này thỏa mãn tiêu chí về kích thước mỗi file (không quá **100MB**).

*Ba vùng này hợp thành khu vực* ***Siêu dữ liệu của hệ thống****. Trong thực tế, người ta chỉ sử dụng 10% kích thước của một ổ cứng để lưu toàn bộ phần siêu dữ liệu. Phần còn lại người ta dùng để lưu dữ liệu.*

1. Vùng **DATA**: Chứa dữ liệu của các file. Dữ liệu lưu thành nhiều sector. Có một số điểm cần chú ý như sau:

* Các sector khác nhau có thể lưu dữ liệu khác nhau. Dựa vào thuộc tính **“Danh sách các sector”** ở vùng **ENTRY**, người ta có thể truy xuất lần lượt từng vị trí của các sector và ghép thành file.
  + ***Ví dụ:*** *Một file có kích thước 1028 bytes lưu theo thứ tự 1026, 1024, 1027 sẽ được đọc lên lần lượt là****: Byte 0 – 511 đọc từ sector 1026, byte 512 – 1023 đọc từ sector 1024 và còn lại đọc từ sector 1027.***
* Đối với thư mục, cấu trúc bên trong sẽ là 32 phần bằng nhau, mỗi phần là 16 bytes, có 2 thuộc tính:

|  |  |  |
| --- | --- | --- |
| **Cộng dồn** | **Độ lớn** | **Nội dung** |
| 0 | 14 | Tên file/Thư mục |
| 14 | 2 | Mã entry nối tiếp. |

* Để truy xuất các file thuộc thư mục con (Giả sử file **‘/data/hcmus/tuimetqua.txt’** chẳng hạn), ta sẽ làm như sau:
  + **Bước 1**: Tìm mã Entry của “/”, thường là Entry 1.
  + **Bước 2**: Tìm trong nội dung sector được Entry 1 quản lý cụm từ **“data”**, giả sử nhảy tới Entry khớp kế tiếp là Entry 7
  + **Bước 3**: Tìm trong nội dung sector được Entry 7 quản lý cụm từ **“hcmus”**, giả sử nhảy tới Entry khớp kế tiếp là Entry 10
  + **Bước 4**: Tìm trong nội dung sector được Entry 10 quản lý. Tìm thấy file “tuimetqua.txt”.

|  |  |  |
| --- | --- | --- |
| **INFO** | **ENTRY** | **DATA** |

1. Do khó khăn trong quá trình triển khai mô hình ở câu 1, nhóm quyết định chuyển sang mô hình thứ hai với 3 vùng quản lý như sau:

Cụ thể mỗi vùng sẽ có những thông số kỹ thuật sau:

|  |  |  |
| --- | --- | --- |
| **Offset** | **Độ lớn** | **Nội dung** |
| 0 | 4 | Thông tin file (mặc định giá trị) |
| 4 | 4 | Kích thước vùng Entry (tính theo byte) |
| 8 | 4 | Vị trí bắt đầu vùng Entry (tính từ đầu file Volume) |

1. Vùng **INFO:** Vùng này có nhiệm vụ sẽ lưu toàn bộ thông tin của một Volume. Vùng này sẽ được đọc lên trước ngay khi mở file để lấy những thông tin cơ bản của Volume. Cấu trúc của vùng này có những thuộc tính sau (thông số kỹ thuật tính theo đơn vị byte):

Điều cần chú ý về file này:

* Thông tin file cần được gán giá trị mặc định, nhằm phân biệt với các loại file khác. Giá trị này được nhóm gán mặc định là: **0x03D1B59B**
* Với hai thông tin nằm ở hai offset số 4 (Kích thước vùng Entry) và 8 (Vị trí bắt đầu vùng Entry), ta sẽ thực hiện truy xuất vùng Entry (tương đương với một con trỏ file)

1. Vùng **ENTRY**: Vùng này có cấu trúc của một bảng các Entry. Mỗi Entry sẽ tương ứng với một file con hoặc thư mục con bên trong Volume (khá giống một con trỏ file, giúp quản lý chính xác 1 file hoặc thư mục trong Volume).

Cấu trúc của vùng này có những thuộc tính sau (thông số kỹ thuật tính theo byte):

|  |  |  |
| --- | --- | --- |
| **Offset** | **Độ lớn** | **Nội dung** |
| 0 | 3 | Thời gian chỉnh sửa lần cuối |
| 3 | 2 | Ngày chỉnh sửa lần cuối |
| 5 | 4 | Kích thước dữ liệu cho mỗi file/thư mục ở vùng DATA |
| 9 | 2 | Độ dài tên file/folder (n) |
| 11 | 2 | Độ dài mật khẩu (m) |
| 13 | 4 | Vị trí dữ liệu (tính từ đầu Volume) |
| 17 | n | Tên file/thư mục |
| 17+n | m | Mật khẩu của file/thư mục |

Các yếu tố cần chú ý như sau:

* **Vị trí file/thư mục**: Kết hợp việc đọc kích thước file/thư mục ở trên sẽ giúp ta có được con trỏ file và truy xuất chính xác nội dung vùng DATA.
* **Tên file/thư mục**: Độ dài n, truy xuất từ thuộc tính “Độ dài tên file”. Với kích thước 2 bytes tại thuộc tính độ dài, như vậy, một file/thư mục sẽ có chiều dài tối đa là (216 – 1), tức là 65535. Một file/thư mục thực tế không thể nào có độ dài lớn đến như vậy. Cách lưu của một file/thư mục cũng giống như cách lưu ở trên Hệ điều hành (ví dụ: “**D:/Data/tuimetqua.txt**” chẳng hạn). Nếu như lưu đường dẫn thư mục, ta sẽ lưu đến dấu “/”. Đây chính là dấu chỉ thị, giúp xác định cấp thư mục để thuận lợi cho việc truy xuất.

1. Vùng **DATA**: Chứa dữ liệu của các file. Thực hiện theo cơ chế FIFO (First In/First Out), dữ liệu sẽ liên tiếp nhau mà không cần chỉ thị bắt đầu/kết thúc file. Một điểm cần chú ý đó chính là: Với cơ chế FIFO, file/thư mục con sẽ được xếp sau file/thư mục cha (không cần liền kề, bởi vì đã có thuộc tính “Tên file/thư mục” quyết định).

Một số đoạn mã nguồn và giải thích thuật toán sẽ có ở phần dưới:

* + - 1. **Tạo mới 1 Volume**Ảnh có chứa văn bản

         Mô tả được tạo tự động

Thuật toán được cụ thể như sau:

* **Bước 1**: Nhập đường dẫn tại nơi chứa Volume File. Kiểm tra tính hợp lệ của đường dẫn này, nếu không hợp lệ thì thông báo lỗi, kết thúc chức năng. Nếu hợp lệ chuyển sang bước 2.
* **Bước 2**: Chép các nội dung sau đây vào vùng Volume Info, sau đó kết thúc chức năng:
  + Định danh: 0x03D1B59B
  + Kích thước vùng ENTRY TABLE = 0
  + Offset bắt đầu vùng = 0
    - 1. **Mở 1 VolumeẢnh có chứa văn bản

         Mô tả được tạo tự động**

Thuật toán được cụ thể như sau:

* **Bước 1**: Nhập đường dẫn Volume File. Kiểm tra tính hợp lệ của đường dẫn, nếu không hợp lệ thì kết thúc chức năng, ngược lại thì chuyển sang bước tiếp theo.
* **Bước 2**: Kiểm tra định dạng Volume File.
  + Bước 2a: Dời con trỏ về cuối Volume File, sau đó đọc về 12 bytes.
  + Bước 2b: Kiểm tra vùng Volume Info. Nếu không đúng định dạng thì kết thúc chức năng. Ngược lại sang bước tiếp theo.
* **Bước 3**: Đọc vùng Volume Info. Nếu chưa có trong Cache thì lưu vào Cache
* **Bước 4**: Đọc tiếp nội dung của vùng Entry Table. Sau đó đọc từng Entry vào RAM theo cấu trúc tuần tự.
  + - 1. **Import tập tin/thư mục từ bên ngoài vào Volume**

Thuật toán thực hiện như sau:

* **Bước 1**: Phân tích kiểu import (Tập tin hay thư mục)
* **Bước 2a (Nếu là tập tin)**: Lấy thông tin và tạo Entry mới ở cuối vùng Entry Table để lưu và quản lý. Ta chép dữ liệu của tập tin vào vùng Data, chuyển sang bước 3.
* **Bước 2b (Nếu là thư mục)**: Tiến hành duyệt lần lượt tất cả các tập tin và thư mục con trong thư mục cần import. Mỗi khi truy cập, ta lưu thông tin vào 1 Entry nằm trong vùng Entry Table. Nếu là tập tin, chép tập tin vào vùng Data. Lưu ý: Nếu dữ liệu đi vào giữa Volume, phải di chuyển dữ liệu về phai sau.
* **Bước 3**: Cập nhật kích thước và offset của vùng Entry Table để lưu trong Volume Info:
  + - 1. **Đặt Mật khẩu**

Ảnh có chứa văn bản

Mô tả được tạo tự độngĐây là một tính năng được nâng cao để phục vụ cho việc mã hóa tập tin/thư mục. Nhóm chủ trương sử dụng thuật toán băm SHA-256, được bảo vệ 3 lớp: **Lớp Salt**, **Lớp Pepper** và **Lớp SHA-256**. Với việc bảo vệ 3 lớp này, gần như rất khó cho tin tặc có thể tấn công để chiếm quyền truy cập thành công.

Mã nguồn của cơ chế SHA256, nhóm sử dụng nguyên vẹn trong 2 tập tin **SHA256.h** và **SHA256.cpp,** được ghi nhận tại mục 2 của phần [tài liệu tham khảo](#_TÀI_LIỆU_THAM) ở cuối bản báo cáo này.

Ảnh bên trái là cơ chế **addSalt** (mã hóa ngẫu nhiên 3 ký tự bằng phép XOR) và **addPepper** (thêm ngẫu nhiên 1 ký tự từ bảng ASCII) vào mật khẩu.

* + - 1. **Export tập tin/thư mục từ Volume ra bên ngoài**

Thuật toán được cụ thể hóa như sau:

* **Bước 1**: Phân tích kiểu export (Tập tin hay thư mục)
* **Bước 2a** (Nếu là tập tin): Tạo tập tin trống tại vị trí muốn export. Chép dữ liệu từ tập tin ra bên ngoài. Kết thúc quá trình.
* **Bước 2b** (Nếu là thư mục): Duyệt cây thư mục, lần lượt duyệt từng Entry. Nếu là thư mục con của thư mục cần export thì tiếp tục truy xuất vào bên trong để nhằm xóa các tập tin trong Volume, đồng thời chép dữ liệu tập tin từ Volume ra bên ngoài. Kết thúc quá trình.
  + - 1. **Xóa 1 tập tin/thư mục trong Volume**

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự độngẢnh có chứa văn bản

Mô tả được tạo tự động

Thuật toán được cụ thể như sau:

* **Bước 1**: Tìm tất cả tập tin và thư mục con của thư mục cần xóa. Nếu là tập tin thì chuyển sang bước 2.
* **Bước 2**: Kiểm tra xem tập tin có bị khóa hay không, nếu bị khóa thì bỏ qua.
* **Bước 3**: Tiến hành xóa tập tin/thư mục theo thứ tự sau:
  + 3a. Dời nội dung của DATA lên ngay chính nội dung cần xóa
  + 3b. Xóa Entry quản lý file ra khỏi EntryTable. Cập nhật vị trí bắt đầu dữ liệu của các Entry đứng sau.
  + 3c. Viết xuống vùng Volume File toàn bộ dữ liệu Entry Table.
  + 3d. Cập nhật Volume Info. Yêu cầu hệ điều hành cập nhật kích thước Volume File (điều này là cực kỳ quan trọng vì nếu không sẽ làm hỏng cấu trúc file).

***Đánh giá chung***: Do gặp một số trục trặc nên việc triển khai giao diện chưa thật sự thành công. Tuy nhiên theo như yêu cầu đã đạt được khoảng 70-80%. Một số tính năng chưa thật sự hoàn hảo, đặc biệt là tính năng xóa tập tin/thư mục. Nhóm cũng chưa thể triển khai việc đọc volume theo cơ chế cluster/sector, do đó việc này cũng là hạn chế cần khắc phục trong tương lai.

# TÀI LIỆU THAM KHẢO

1. Lecture 03: Layering, Naming & Filesystem Design, Chris Gregg, CS110 – Principles of Computer Systems, Stanford University (2019): <https://web.stanford.edu/class/archive/cs/cs110/cs110.1196/static/lectures/03-Layering-Naming-Filesystem-Design/lecture-03-layering-naming-filesystem-design.pdf>
2. Mã nguồn file **SHA256.h** và **SHA256.cpp**:

***Copyright © 2014, 2015 Stephan Brumme. All rights reserved.***

1. Slides bài giảng CDIO Hệ điều hành – Khoa Công nghệ thông tin, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia TPHCM.