

## Giới thiệu

Trang này nhằm mục đích giới thiệu về hệ thống tập Bảng phân bổ tập (FAT) gốc. Hệ thống tập này được sử dụng trên tất cả các phiên bản MS-DOS và PC-DOS cũng như trên các phiên bản Windows đầu tiên; nó vẫn được sử dụng trên các đĩa mềm được định dạng bởi Windows và một số hệ thống khác. Các phiên bản sửa đổi vẫn được Windows hỗ trợ trên đĩa cứng, nếu được yêu cầu.

Hệ thống tập FAT chủ yếu dựa vào mô hình *bản đồ tập* về mặt bố cục trên đĩa; mô hình đó đã tồn tại trong nhiều năm trước khi Microsoft kế thừa hệ thống tập FAT ban đầu từ những người viết ban đầu của DOS (Sản phẩm máy tính Seattle). Nó là một hệ thống tập tin khá đơn giản và mạnh mẽ.

Có ba biến thể cơ bản của hệ thống tập FAT, chúng khác nhau chủ yếu ở cách xây dựng bảng phân bổ tập thực tế. Đĩa mềm và đĩa cứng nhỏ thường sử dụng phiên bản *12 bit*, phiên bản này đã được thay thế bằng phiên bản *16 bit* khi đĩa cứng trở nên lớn hơn. Đến lượt phiên bản này đã được thay thế bởi phiên bản *32-bit* khi đĩa ngày càng trở nên lớn hơn. Chúng ta sẽ tập trung vào phiên bản 16 bit, vì phiên bản 12 bit có thể khó đối với người mới bắt đầu và phiên bản 32 bit phức tạp hơn mức cần thiết cho hướng dẫn này.

## Tổng quan

Bất kỳ đĩa nào cũng được tạo thành từ *các bề mặt* (một bề mặt cho mỗi đầu), *các rãnh* và *các cung*. Tuy nhiên, để đơn giản, chúng ta có thể coi đĩa là một vùng lưu trữ đơn giản chỉ được tạo thành từ một số lĩnh vực. Hơn nữa, các lĩnh vực này được coi là được đánh số liên tiếp, lĩnh vực đầu tiên được đánh số 0, lĩnh vực thứ hai được đánh số 1, v.v.; chúng tôi sẽ không lo lắng về vị trí vật lý của bất kỳ khu vực nào trên đĩa thực. Bởi vì chúng muốn nhấn mạnh rằng vị trí của một khu vực không liên quan đến cấu trúc đĩa thực tế và bởi vì các khu vực có số riêng trong mỗi rãnh ghi, nên từ bây giờ chúng tôi sẽ gọi *các khối khu vực này là các khối khu vực*; như đã nêu trước đây, chúng tạo thành một danh sách tuyến tính, được đánh số dày đặc.

Tất cả các khối đều có cùng kích thước, 512 byte, trên thực tế tất cả các hệ thống tập FAT. Tuy nhiên, các đĩa lớn có thể có quá nhiều khối để tạo sự thoải mái, vì vậy các khối đôi khi được nhóm lại với nhau theo cặp (hoặc bốn, hoặc tám, v.v...); mỗi nhóm như vậy được gọi là *đơn vị phân bổ*. Hệ thống tập FAT thực sự hoạt động theo đơn vị phân bổ, không phải khối, nhưng để đơn giản, chúng tôi giả định trong phần mô tả bên dưới rằng mỗi đơn vị phân bổ chứa chính xác một khối, có nghĩa là chúng tôi có thể sử dụng các thuật ngữ thay thế cho nhau.

## Lưu ý về giá trị số

Các số thập lục phân được biểu thị bằng cách sử dụng quy ước thường được sử dụng trong C; tức là dẫn đầu 0x. Số thập phân 17 do đó sẽ được viết là 0x11 trong ký hiệu thập lục phân ở đây.

Các giá trị trong hệ thống tập FAT được lưu trữ theo *byte* (giá trị 8 bit, 0-255 không dấu) hoặc bằng *từ* (cặp byte, giá trị 16 bit, 0-65535 không dấu). Lưu ý rằng *byte đầu tiên của một cặp là byte có trọng số thấp nhất và byte thứ hai của một cặp là byte có trọng số cao nhất*. Ví dụ: nếu byte ở vị trí 3 có giá trị 0x15 và byte ở vị trí 4 có giá trị 0x74 thì chúng cùng nhau tạo thành một từ có giá trị 0x7415 (không phải 0x1574).

Thỉnh thoảng có các giá trị 32 bit ( *doublewords* ) và các giá trị này sử dụng cách tiếp cận tương tự (trong trường hợp này là 4 byte, với byte ít quan trọng nhất được lưu trước tiên).

Cuối cùng, lưu ý rằng *các bit riêng lẻ trong một byte hoặc từ được đánh số từ đầu ít quan trọng nhất* (đầu bên phải), *bắt đầu bằng bit 0*.

## Định dạng đĩa

Phần này mô tả *cấu trúc trên đĩa* của hệ thống tập tin FAT; nghĩa là các vùng khác nhau của đĩa được sắp xếp như thế nào và những gì được lưu trữ trong đó.

### Bố cục cơ bản

Tất cả các đĩa sử dụng hệ thống tập tin FAT được chia thành nhiều khu vực. Bảng sau đây tóm tắt các vùng theo thứ tự xuất hiện trên đĩa, bắt đầu từ khối 0:

Mô tả khu vực	Kích thước diện tích
<a href="#">Khởi khởi động</a>	1 khối
<a href="#">Bảng phân bổ tệp</a> (có thể có nhiều bản sao)	Phụ thuộc vào kích thước hệ thống tập tin
<a href="#">Thư mục gốc</a> đĩa	Biến (được chọn khi đĩa được định dạng)
Vùng dữ liệu tập tin	Phần còn lại của đĩa

### Khởi khởi động

**Khởi khởi động chỉ chiếm khối đầu tiên của đĩa.** Nó chứa một chương trình đặc biệt ( *chương trình khởi động* ) được sử dụng để tải hệ điều hành vào bộ nhớ. Do đó, nó có vẻ khá không liên quan đến cuộc thảo luận này.

Tuy nhiên, trong hệ thống tập tin FAT, nó cũng chứa một số **vùng dữ liệu** quan trọng giúp **mô tả phần còn lại của hệ thống tệp**. Vì vậy, để hiểu cách bố trí một đĩa cụ thể, trước tiên cần phải hiểu ít nhất một phần nội dung của khối khởi động. Các khu vực liên quan được hiển thị trong bảng sau, cùng với độ lệch byte của chúng từ đầu khối khởi động. Sau này chúng ta sẽ thấy điều nào trong số này thực sự quan trọng đối với chúng ta.

Bù đắp từ đầu	Chiều dài	Sự miêu tả
0x00	3 byte	Một phần của chương trình khởi động.
0x03	8 byte	Mô tả nhà sản xuất tùy chọn.
0x0b	2 byte	Số byte trên mỗi khối (hầu như luôn luôn là 512).
0x0d	1 byte	Số khối trên mỗi đơn vị phân bổ.
0x0e	2 byte	Số khối dành riêng. Đây là số khối trên đĩa không thực sự là một phần của hệ thống tệp; trong hầu hết các trường hợp, đây chính xác là 1, là mức cho phép cho khối khởi động.
0x10	1 byte	Số lượng <a href="#">bảng phân bổ tệp</a> .
0x11	2 byte	Số lượng mục nhập <a href="#">thư mục gốc</a> (bao gồm cả những mục không sử dụng).
0x13	2 byte	Tổng số khối trong toàn bộ đĩa. Nếu kích thước đĩa lớn hơn 65535 khối (và do đó sẽ không vừa với hai byte này), giá trị này được đặt thành 0 và kích thước thực được lưu ở <a href="#">offset 0x20</a> .
0x15	1 byte	<a href="#">Bộ mô tả phương tiện truyền thông</a> . Điều này hiếm khi được sử dụng, nhưng vẫn tồn tại. .
0x16	2 byte	Số khối được chiếm bởi một bản sao của <a href="#">Bảng phân bổ tệp</a> .

0x18	2 byte	Số khối trên mỗi rãnh. Thông tin này chủ yếu được cung cấp cho việc sử dụng chương trình khởi động và chúng tôi không cần phải quan tâm thêm ở đây.
0x1a	2 byte	Số lượng đầu (bề mặt đĩa). Thông tin này chủ yếu được cung cấp cho việc sử dụng chương trình khởi động và chúng tôi không cần phải quan tâm thêm ở đây.
0x1c	4 byte	Số lượng <i>khối ẩn</i> . Việc sử dụng điều này phần lớn mang tính lịch sử và gần như luôn được đặt thành 0; do đó nó có thể được bỏ qua.
0x20	4 byte	Tổng số khối trong toàn bộ đĩa (xem thêm <a href="#">offset 0x13</a> ).
0x24	2 byte	Số ổ đĩa vật lý. Thông tin này chủ yếu được cung cấp cho việc sử dụng chương trình khởi động và chúng tôi không cần phải quan tâm thêm ở đây.
0x26	1 byte	Chữ ký bản ghi khởi động mở rộng Thông tin này chủ yếu được cung cấp cho việc sử dụng chương trình khởi động và chúng tôi không cần phải quan tâm thêm ở đây.
0x27	4 byte	Số sê-ri tập. Số duy nhất được sử dụng để nhận dạng một đĩa cụ thể.
0x2b	11 byte	Nhãn khối lượng. Đây là một chuỗi ký tự để nhận dạng đĩa mà con người có thể đọc được (được đệm bằng dấu cách nếu ngắn hơn); nó được chọn khi đĩa được định dạng.
0x36	8 byte	Mã định danh hệ thống tệp (được đệm ở cuối bằng dấu cách nếu ngắn hơn).
0x3e	0x1c0 byte	Phần còn lại của chương trình khởi động.
0x1fe	2 byte	Khối khởi động 'chữ ký' (0x55 theo sau là 0xaa).

## Bộ mô tả phương tiện truyền thông

Trong lịch sử, hệ điều hành khó xác định được kích thước và loại đĩa chỉ bằng cách thăm vắn phần cứng. Do đó, một 'byte ma thuật' đã được sử dụng để phân loại đĩa. Điều này vẫn còn tồn tại nhưng hiếm khi được sử dụng và nội dung của nó được gọi là Bộ mô tả phương tiện. Nói chung, đối với đĩa cứng, giá trị này được đặt thành 0xf0.

## Bảng phân bổ tệp (FAT)

FAT chiếm một hoặc nhiều khối ngay sau khối khởi động. Thông thường, một phần của khối cuối cùng của nó sẽ không được sử dụng, vì không chắc số lượng mục được yêu cầu sẽ lấp đầy chính xác số khối hoàn chỉnh. Nếu có FAT thứ hai, thì FAT này ngay sau FAT đầu tiên (nhưng bắt đầu ở một khối mới). Điều này được lặp lại cho bất kỳ FAT nào nữa.

Lưu ý rằng nhiều FAT được sử dụng đặc biệt trên các đĩa mềm vì khả năng xảy ra lỗi khi đọc đĩa cao hơn. Nếu FAT không thể đọc được, các tập tin không thể truy cập được và phải sử dụng một bản sao FAT khác. Trên đĩa cứng thường chỉ có một FAT.

Trong trường hợp hệ thống tệp FAT 16 bit, mỗi mục trong FAT có độ dài hai byte (tức là 16 bit). Vùng dữ liệu đĩa được chia thành *các cụm*, tương tự như các đơn vị phân bổ, nhưng được đánh số khác nhau (thay vì được đánh số từ đầu đĩa, chúng được đánh số từ đầu vùng dữ liệu đĩa). Vì vậy, số cụm là số đơn vị phân bổ, trừ đi một giá trị không đổi là kích thước của các vùng ở giữa điểm bắt đầu của đĩa và điểm bắt đầu của vùng dữ liệu.

**Vâng, gần như vậy. Các cụm được đánh số bắt đầu từ 2, không phải 0! Vì vậy, phép tính ở trên phải thêm 2 vào để lấy số cụm của một đơn vị phân bổ nhất định...và số cụm được chuyển thành số đơn vị phân bổ bằng cách trừ đi 2...!**

Vậy FAT hoạt động như thế nào? Nói một cách đơn giản, có một mục nhập trong FAT cho mỗi cụm (khối vùng dữ liệu) trên đĩa. Mục N liên quan đến cụm N. Cụm 0 và 1 không tồn tại (do 'fiddle by 2' ở trên) và các mục FAT đó là đặc biệt. Byte đầu tiên của mục nhập đầu tiên là bản sao của byte [mô tả phương tiện](#) và byte thứ hai được đặt thành 0xff. Cả hai byte trong mục nhập thứ hai được đặt thành 0xff.

Mục nhập FAT bình thường cho một cụm chứa gì? Nó chứa *số cụm kế tiếp* - nghĩa là số cụm theo sau cụm này trong tệp mà cụm hiện tại thuộc về. Cụm cuối cùng của tệp có giá trị 0xffff trong mục FAT để cho biết rằng không còn cụm nào nữa.

## Thư mục gốc

Thư mục gốc chứa một mục nhập cho mỗi tệp có tên xuất hiện ở thư *mục gốc* (cấp cao nhất) của hệ thống tệp. Các thư mục khác có thể xuất hiện trong thư mục gốc; chúng được gọi là *thư mục con*. Sự khác biệt chính giữa hai loại này là không gian dành cho thư mục gốc được phân bổ tĩnh khi đĩa được định dạng; do đó có giới hạn trên hữu hạn về số lượng tệp có thể xuất hiện trong thư mục gốc.

Thư mục con chỉ là các tệp có dữ liệu đặc biệt trong đó, vì vậy chúng có thể lớn hoặc nhỏ tùy ý.

Định dạng của tất cả các thư mục đều giống nhau. Mỗi mục có kích thước 32 byte (0x20), vì vậy một khối có thể chứa 16 mục trong số đó. Bảng sau đây trình bày tóm tắt về một mục nhập thư mục; lưu ý rằng phần bù chỉ tính từ đầu mục cụ thể đó, không phải từ đầu khối.

Bù lại	Chiều dài	Sự miêu tả
0x00	8 byte	<a href="#">Tên tệp</a>
0x08	3 byte	<a href="#">Phân mở rộng tên tệp</a>
0x0b	1 byte	<a href="#">Thuộc tính tệp</a>
0x0c	10 byte	Kín đáo
0x16	2 byte	<a href="#">Thời gian tạo hoặc cập nhật lần cuối</a>
0x18	2 byte	<a href="#">Ngày tạo hoặc cập nhật lần cuối</a>
0x1a	2 byte	<a href="#">Số cụm bắt đầu</a> cho tệp
0x1c	4 byte	<a href="#">Kích thước tệp tính bằng byte</a>

## Tên tệp

Tám byte từ offset 0x00 đến 0x07 đại diện cho tên tệp. Byte đầu tiên của tên tệp cho biết trạng thái của nó. Thông thường, nó chứa ký tự tên tệp thông thường (ví dụ: 'A'), nhưng có một số giá trị đặc biệt:

- 0x00  
Tên tập tin không bao giờ được sử dụng.
- 0xe5  
Tên tệp đã được sử dụng nhưng tệp đã bị xóa.
- 0x05  
Ký tự đầu tiên của tên tệp thực sự là 0xe5.
- 0x2e

Mục nhập dành cho một thư mục, không phải một tập tin bình thường. Nếu byte thứ hai cũng là 0x2e thì trường cụm chứa số cụm của thư mục mẹ của thư mục này. Nếu thư mục mẹ là thư mục gốc (được phân bổ tĩnh và không có số cụm), số cụm 0x0000 được chỉ định ở đây.

Bất kỳ nhân vật nào khác

Đây là ký tự đầu tiên của tên tệp thực.

Nếu tên tệp có độ dài ít hơn tám ký tự, nó sẽ được đệm bằng các ký tự khoảng trắng.

## Phần mở rộng tên tệp

Ba byte từ offset 0x08 đến 0x0a biểu thị phần mở rộng tên tệp. Không có ký tự đặc biệt. Lưu ý rằng dấu chấm được sử dụng để phân tách tên tệp và phần mở rộng tên tệp được ngụ ý và không thực sự được lưu trữ ở bất kỳ đâu; nó chỉ được sử dụng khi đề cập đến tập tin. Nếu phần mở rộng tên tệp có độ dài ít hơn ba ký tự, nó sẽ được đệm bằng các ký tự khoảng trắng.

## Thuộc tính tệp

Byte đơn ở offset 0x0b chứa các cờ cung cấp thông tin về tệp và các quyền của nó, v.v. Các cờ là các bit đơn và có ý nghĩa như sau. Mỗi bit được đưa ra dưới dạng giá trị số và chúng được kết hợp để tạo ra giá trị thuộc tính thực tế:

- 0x01 Cho biết tập tin ở chế độ chỉ đọc.
- 0x02 Cho biết một tập tin ẩn. Những tập tin như vậy có thể được hiển thị nếu nó thực sự cần thiết.
- 0x04 Cho biết một tập tin hệ thống. Những điều này cũng được ẩn đi.
- 0x08 Chỉ ra một mục đặc biệt chứa nhãn ổ đĩa, thay vì mô tả một tập tin. Loại mục này chỉ xuất hiện trong thư mục gốc.
- 0x10 Mục này mô tả một thư mục con.
- 0x20 Đây là cờ lưu trữ. Điều này có thể được thiết lập và xóa bởi lập trình viên hoặc người dùng, nhưng luôn được thiết lập khi tệp được sửa đổi. Nó được sử dụng bởi các chương trình sao lưu.
- 0x40 Không được sử dụng; phải được đặt thành 0.
- 0x80 Không được sử dụng; phải được đặt thành 0.

## Thời gian tập tin

Hai byte ở độ lệch 0x16 và 0x17 được coi là giá trị 16 bit; hãy nhớ rằng byte có ý nghĩa nhỏ nhất nằm ở offset 0x16. Chúng chứa thời gian khi tệp được tạo hoặc cập nhật lần cuối. Thời gian được ánh xạ theo bit như sau; dòng đầu tiên biểu thị độ lệch của byte, dòng thứ hai biểu thị số bit riêng lẻ (ở dạng thập phân) trong giá trị 16 bit và dòng thứ ba cho biết nội dung được lưu trữ trong mỗi bit.

```
<----- 0x17 -----> <----- 0x16 ----->
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
hhhhmmmmmmxxxxxx
```

Ở đâu:

hhh

cho biết số giờ nhị phân (0-23)

mmmmmm

cho biết số phút nhị phân (0-59)

xxxxxx

cho biết số nhị phân của khoảng thời gian hai giây (0-29), biểu thị số giây từ 0 đến 58.

## Ngày tập tin

Hai byte ở độ lệch 0x18 và 0x19 được coi là giá trị 16 bit; hãy nhớ rằng byte có ý nghĩa nhỏ nhất nằm ở offset 0x18. Chúng chứa ngày tập được tạo hoặc cập nhật lần cuối. Ngày được ánh xạ theo các bit như sau; dòng đầu tiên biểu thị độ lệch của byte, dòng thứ hai biểu thị số bit riêng lẻ (ở dạng thập phân) trong giá trị 16 bit và dòng thứ ba cho biết nội dung được lưu trữ trong mỗi bit.

```
<----- 0x19 -----> <----- 0x18 ----->
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
yyyyyyymmmddddd
```

Ở đâu:

yyyyyy

biểu thị phần bù năm nhị phân từ 1980 (0-119), đại diện cho các năm 1980 đến 2099

mmmm

cho biết số tháng nhị phân (1-12)

dddd

cho biết số ngày nhị phân (1-31)

## Số cụm bắt đầu

Hai byte ở độ lệch 0x1a và 0x1b được coi là giá trị 16 bit; hãy nhớ rằng byte có ý nghĩa nhỏ nhất nằm ở offset 0x1a. Cụm đầu tiên dành cho không gian dữ liệu trên đĩa luôn được đánh số là 0x0002. Sự sắp xếp kỳ lạ này là do hai mục đầu tiên trong FAT được dành riêng cho các mục đích khác.

## Kích thước tập tin

Bốn byte ở độ lệch 0x1c đến 0x1f được coi là giá trị 32 bit; hãy nhớ rằng byte có ý nghĩa nhỏ nhất nằm ở offset 0x1c. Chúng giữ kích thước tệp thực tế, tính bằng byte.

## Ví dụ đã làm việc

Cách tốt nhất để hiểu cách sử dụng thông tin trên là làm việc qua một số ví dụ đơn giản.

## Giải thích nội dung của một khối

Chúng tôi giả định rằng có một công cụ có sẵn để hiển thị nội dung của một khối ở cả ký tự thập lục phân và ký tự ASCII. Hầu hết các công cụ như vậy sẽ hiển thị các ký tự ASCII bất thường (ví dụ như dấu xuống dòng) dưới dạng dấu chấm. Ví dụ: đây là màn hình hiển thị khối khởi động điển hình:



Block 0 (0x0000)																
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	eb	3c	90	49	42	4d	2d	37	2e	30	20	00	02	01	01	00
010	01	40	00	a1	13	f8	14	00	0a	00	01	00	00	00	00	00
020	00	00	00	00	00	00	29	2a	65	bc	00	43	4f	38	38	33
030	2d	41	32	20	20	20	46	41	54	31	36	20	20	20	fa	31
040	c0	8e	d0	bc	00	7c	fb	8e	d8	e8	00	00	5e	83	c6	19
050	bb	07	00	fc	ac	84	c0	74	06	b4	0e	cd	10	eb	f5	30
060	e4	cd	16	cd	19	0d	0a	4e	6f	6e	2d	73	79	73	74	65
070	6d	20	64	69	73	6b	0d	0a	50	72	65	73	73	20	61	6e
080	79	20	6b	65	79	20	74	6f	20	72	65	62	6f	6f	74	0d
090	0a	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	aa

Như minh họa, một trường trong khối khởi động đã được đánh dấu màu đỏ (điểm đánh dấu xuất hiện hai lần, một lần cho biểu diễn thập lục phân và một lần cho biểu diễn ASCII). Các số ở phía bên trái là độ lệch (từ đầu khối) của byte đầu tiên trên hàng đó và hàng chữ số đầu tiên dọc theo đầu là độ lệch của mỗi byte trong hàng. Do đó, chúng ta có thể dễ dàng thấy rằng vùng được đánh dấu bắt đầu ở offset 0x36.

Khu vực được đề cập là (xem lại bố cục khối khởi động) loại hệ thống tệp, trong trường hợp này là FAT16. Để giúp chúng ta dễ dàng tra cứu từng byte trong bảng ký tự ASCII, chúng ta chỉ cần tham khảo cách biểu diễn tương đương ở phía bên phải. 0x46 đại diện cho F, 0x41 đại diện cho A, v.v.

## Ví dụ 1 - tìm thư mục gốc

Để tìm thư mục gốc, chúng ta cần kiểm tra dữ liệu hệ thống tệp trong khối khởi động. Vì vậy, chúng ta hãy nhìn lại khối khởi động của đĩa ví dụ của chúng ta:

Block 0 (0x0000)																
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	eb	3c	90	49	42	4d	2d	37	2e	30	20	00	02	01	01	00
010	01	40	00	a1	13	f8	14	00	0a	00	01	00	00	00	00	00
020	00	00	00	00	00	00	29	2a	65	bc	00	43	4f	38	38	33
030	2d	41	32	20	20	20	46	41	54	31	36	20	20	20	fa	31
040	c0	8e	d0	bc	00	7c	fb	8e	d8	e8	00	00	5e	83	c6	19
050	bb	07	00	fc	ac	84	c0	74	06	b4	0e	cd	10	eb	f5	30
060	e4	cd	16	cd	19	0d	0a	4e	6f	6e	2d	73	79	73	74	65
070	6d	20	64	69	73	6b	0d	0a	50	72	65	73	73	20	61	6e
080	79	20	6b	65	79	20	74	6f	20	72	65	62	6f	6f	74	0d
090	0a	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	aa

Chúng tôi biết rằng thư mục gốc xuất hiện ngay sau bản sao cuối cùng của FAT. Vì vậy, điều chúng ta cần tìm hiểu là kích thước của FAT và có bao nhiêu bản sao. Chúng ta cũng cần biết kích thước của bất kỳ thứ gì khác xuất hiện trước (các) FAT; chỉ có một khối duy nhất của khối khởi động. Vì vậy, số khối xuất hiện trước thư mục gốc được cho bởi:

$$(\text{kích thước của FAT}) * (\text{số lượng FAT}) + 1$$

Khi đó, tất cả những gì chúng ta cần làm là khám phá những giá trị này. Đầu tiên, chúng ta biết rằng số lượng FAT được lưu trữ ở offset 0x10 (được đánh dấu bằng màu xanh lá cây ở trên); điều này cho chúng ta biết rằng chỉ có một FAT. Tiếp theo, chúng ta cần biết kích thước của FAT; đây là ở các độ lệch 0x16 và 0x17, trong đó chúng tôi tìm thấy 0x14 và 0x00 tương ứng (được đánh dấu màu đỏ ở trên). Hãy nhớ rằng hai byte này cùng nhau tạo thành một giá trị 16 bit, với byte ít quan trọng nhất được lưu trước tiên; nói cách khác, giá trị là 0x0014 (ở dạng thập phân, 20). Vì vậy, tổng số khối đứng trước thư mục gốc được cho bởi:

$$0x0014 * 1 + 1 \Rightarrow 0x0015 \text{ (21 thập phân)}$$

Do đó, chúng ta nên tìm thư mục gốc trong khối 0x15, vì vậy hãy xem xét nó...



Block 21 (0x0015)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	43	4f	38	38	33	2d	41	32	20	20	20	28	00	00	00	00	C0883-A2 (.....
010	00	00	00	00	00	00	91	9e	65	39	00	00	00	00	00	00	.....e9.....
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Dường như có thứ gì đó chiếm 0x20 byte đầu tiên và đó là...một mục nhập thư mục! Chúng ta sẽ không đi vào chi tiết ở đây, nhưng việc kiểm tra chi tiết các byte đó sẽ cho thấy đó là mục nhập đặc biệt dành cho nhãn đĩa. Đường như không còn mục nào nữa trong thư mục này.

## Ví dụ 2 - tìm thuộc tính của tệp

Trong ví dụ này, tệp FOOBAR.TXT đã được tạo trên cùng một đĩa và nó xuất hiện trong thư mục gốc. Chúng tôi muốn tìm hiểu xem cờ thuộc tính nào được đặt trên tệp.

Đầu tiên chúng ta cần tìm thư mục gốc; chúng ta đã thực hiện điều này trong ví dụ 1. Hãy xem xét nó sau khi FOOBAR.TXT được tạo:

Block 21 (0x0015)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	43	4f	38	38	33	2d	41	32	20	20	20	28	00	00	00	00	C0883-A2 (....
010	00	00	00	00	00	00	91	9e	65	39	00	00	00	00	00	00	.....e9.....
020	46	4f	4f	42	41	52	20	20	54	58	54	21	00	a3	91	9e	FOOBAR TXT!....
030	65	39	65	39	00	00	91	9e	65	39	c6	10	1a	00	00	00	e9e9.....e9.....
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Chúng ta có thể thấy khá dễ dàng rằng mục nhập thư mục thứ hai (mục ở offset 0x20) là dành cho FOOBAR.TXT. Hãy nhớ rằng dấu chấm giữa tên tệp và phần mở rộng tên tệp không thực sự được lưu trữ mà chỉ được ngụ ý. Chúng ta thấy tên tệp (được đánh dấu màu đỏ) và phần mở rộng tên tệp (được đánh dấu màu xanh lam). Chúng tôi biết rằng byte thuộc tính xuất hiện ở offset 0x0b và nó được đánh dấu bằng màu xanh lục ở đây.

Giá trị của byte thuộc tính là 0x21. Chúng ta có thể biểu diễn điều này dưới dạng nhị phân như sau:

0 0 1 0 0 0 0 1

Lấy từng bit riêng biệt và tạo số thập lục phân từ chúng, chúng ta nhận được:

0 0 1 0 0 0 0 0 => 0x20

0 0 0 0 0 0 0 1 => 0x01

[Bảng giá trị thuộc tính](#) của chúng tôi cho thấy rằng 0x20 có nghĩa là 'cờ lưu trữ' được đặt và 0x01 chỉ ra rằng tệp ở chế độ chỉ đọc.

### Ví dụ 3 - tìm ngày của tệp

Ở đây, chúng tôi muốn ngày được đính kèm vào một tệp cụ thể (chỉ có một ngày được giữ lại, đó là ngày tạo hoặc sửa đổi lần cuối). Tệp được đề cập lại là FOOBAR.TXT.

Hãy nhìn lại thư mục gốc một lần nữa; chúng ta đã thực hiện điều này trong ví dụ 2 và thực sự chúng ta đã biết rằng FOOBAR.TXT có một mục nhập thư mục ở offset 0x20:

Block 21 (0x0015)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	43	4f	38	38	33	2d	41	32	20	20	20	28	00	00	00	00	C0883-A2 (....
010	00	00	00	00	00	00	91	9e	65	39	00	00	00	00	00	00	.....e9.....
020	46	4f	4f	42	41	52	20	20	54	58	54	21	00	a3	91	9e	FOOBAR TXT!....
030	65	39	65	39	00	00	91	9e	65	39	c6	10	1a	00	00	00	e9e9....e9.....
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Lần này chúng tôi quan tâm đến ngày của tệp và chúng tôi biết từ [bố cục thư mục gốc](#) của mình rằng đây là giá trị offset 0x18 trong mỗi mục nhập thư mục. Do đó, ngày của FOOBAR.TXT nằm ở offset 0x20+0x18 hoặc 0x38 (được đánh dấu màu đỏ ở trên). Một lần nữa, đây là giá trị 16 bit với byte ít quan trọng nhất được lưu trước tiên. Các byte lần lượt là 0x65 và 0x39, do đó, việc đảo ngược các byte này và đặt chúng lại với nhau sẽ cho giá trị 0x3965.

Bây giờ tất cả những gì chúng ta phải làm là phân tích các thành phần của giá trị này. Một cách dễ dàng trước tiên là chuyển đổi nó thành nhị phân và điều này thậm chí còn dễ dàng hơn nếu chúng ta lấy nó một chữ số thập lục phân mỗi lần:

```

3 9 6 5
| | | |
VVV

```

```
0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1
```

Hãy đẩy tất cả các chữ số lại với nhau:

0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1

Bây giờ chúng ta có thể phân chia chúng một lần nữa theo các ranh giới tương ứng với các thành phần riêng lẻ của ngày, như được xác định trong [định dạng ngày của tệp](#) . Sau đó, chúng tôi chuyển đổi từng phần trở lại số thập phân:

0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1

| | |  
VV

28 11 5  
(năm tháng ngày)

Hãy nhớ rằng năm được tính dựa trên 1980, vì vậy nếu chúng ta cộng 1980 với 28, chúng ta sẽ có 2008. Do đó toàn bộ ngày là ngày 5 tháng 11 năm 2008.

## Ví dụ 4 - tìm khối dữ liệu cho một tệp

Ở đây, chúng tôi muốn tìm ra số khối chứa dữ liệu cho một tệp cụ thể hiện đã được thêm vào đĩa. Tên của tập tin là NETWORK.VRS.

Một lần nữa, chúng ta tìm thấy thư mục gốc. Đây là nội dung mới nhất của nó, sau khi NETWORK.VRS được tạo:



Block 21 (0x0015)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	43	4f	38	38	33	2d	41	32	20	20	20	28	00	00	00	00	C0883-A2 (....
010	00	00	00	00	00	00	91	9e	65	39	00	00	00	00	00	00	.....e9.....
020	46	4f	4f	42	41	52	20	20	54	58	54	21	00	a3	91	9e	FOOBAR TXT!....
030	65	39	65	39	00	00	91	9e	65	39	c6	10	1a	00	00	00	e9e9.....e9.....
040	4e	45	54	57	4f	52	4b	20	56	52	53	20	00	b6	91	9e	NETWORK VRS ....
050	65	39	65	39	00	00	91	9e	65	39	4e 0f	92 06	00	00	00	00	e9e9.....e9N.....
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Lưu ý rằng mục nhập thư mục thứ ba (bắt đầu từ offset 0x40) là dành cho NETWORK.VRS. Chúng tôi biết rằng số cụm bắt đầu cho dữ liệu tệp chiếm các byte ở độ lệch 0x1a và 0x1b trong một mục nhập thư mục cụ thể; do đó, các byte mà chúng tôi muốn có độ lệch 0x5a và 0x5b (chúng tôi vừa thêm 0x40, độ lệch của phần đầu của mục nhập). Những cái này (được đánh dấu bằng màu đỏ) lần lượt chứa 0x4e và 0x0f, đồng thời hãy nhớ rằng byte đầu tiên là byte ít quan trọng nhất, số chúng ta muốn là 0x0f4e. Ngẫu nhiên, bốn byte tiếp theo (được đánh dấu bằng màu xanh lam) là kích thước tệp, một lần nữa, byte có trọng số thấp nhất được đặt đầu tiên. Đây lần lượt là 0x92, 0x06, 0x00, 0x00, tạo thành giá trị 0x00000692. Giá trị này (ở dạng thập phân) là 1682. Vì vậy, tệp này dài **1682** byte.

Hãy xem lại những gì chúng ta biết cho đến nay...

- Cụm bắt đầu của tệp là cụm 0x0f4e.
- Thư mục gốc bắt đầu ở khối 0x15.
- Đơn vị phân bổ đầu tiên bắt đầu ở khối đầu tiên sau thư mục gốc.

Chúng ta còn cần biết gì nữa? Chúng tôi biết thư mục gốc bắt đầu từ đâu nhưng không biết nó kết thúc ở đâu. Vì vậy chúng ta cần kích thước của thư mục gốc, tính bằng khối. Chúng ta hãy nhìn lại khối khởi động:



Block 0 (0x0000)																
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	eb	3c	90	49	42	4d	2d	37	2e	30	20	00	02	01	01	00
010	01	40	00	a1	13	f8	14	00	0a	00	01	00	00	00	00	00
020	00	00	00	00	00	00	29	2a	65	bc	00	43	4f	38	38	33
030	2d	41	32	20	20	20	46	41	54	31	36	20	20	20	fa	31
040	c0	8e	d0	bc	00	7c	fb	8e	d8	e8	00	00	5e	83	c6	19
050	bb	07	00	fc	ac	84	c0	74	06	b4	0e	cd	10	eb	f5	30
060	e4	cd	16	cd	19	0d	0a	4e	6f	6e	2d	73	79	73	74	65
070	6d	20	64	69	73	6b	0d	0a	50	72	65	73	73	20	61	6e
080	79	20	6b	65	79	20	74	6f	20	72	65	62	6f	6f	74	0d
090	0a	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	aa

Những gì chúng ta cần tìm lần này là số lượng mục tối đa trong thư mục gốc; điều này được khắc phục khi đĩa được định dạng. Chúng tôi biết từ bố [cục khối khởi động](#) rằng điều này xuất hiện trong hai byte bắt đầu ở offset 0x11 trong khối khởi động (các byte này được đánh dấu màu đỏ ở trên). Các byte này lần lượt chứa 0x40 và 0x00, do đó (sắp xếp như bình thường) điều này mang lại cho chúng ta giá trị 0x0040 (64 ở dạng thập phân). Vì vậy, có 64 mục thư mục gốc. Chúng tôi biết rằng một mục nhập thư mục chiếm 32 byte, do đó tổng dung lượng mà thư mục gốc chiếm giữ là 64\*32 byte hoặc 2048 byte. Mỗi khối là 512 byte, do đó số khối bị chiếm bởi thư mục gốc là 2048 chia cho 512...tức là 4.

Vì vậy, thư mục gốc bắt đầu ở khối 0x15. Do đó, đơn vị phân bổ đầu tiên bắt đầu ở 0x15+4 hoặc 0x19. Vì vậy, để chuyển đổi số đơn vị phân bổ thành số khối, chúng ta cần thêm giá trị không đổi 0x19. Và để chuyển đổi số cụm (là số xuất hiện trong thư mục gốc) thành số khối, chúng ta cần thêm 0x17, để cho phép độ lệch kỳ lạ đó là 2.

Bây giờ chúng ta biết rằng khối dữ liệu đầu tiên của tệp nằm ở số cụm 0xf4e (xem ở trên). Thêm hằng số mà chúng tôi đã khám phá, chúng tôi thấy rằng đây là số khối 0xf4e+0x17 hoặc 0xf65. Hãy nhìn vào khối 0xf65:

Block 3941 (0x0f65)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	20	20	20	54	77	61	73	20	74	68	65	20	6e	69	67	68	Twas the nigh
010	74	20	62	65	66	6f	72	65	20	73	74	61	72	74	2d	75	t before start-u
020	70	20	61	6e	64	20	61	6c	6c	20	74	68	72	6f	75	67	p and all throug
030	68	20	74	68	65	20	6e	65	74	2c	0a	20	20	20	20	20	h the net,.
040	6e	6f	74	20	61	20	70	61	63	6b	65	74	20	77	61	73	not a packet was
050	20	6d	6f	76	69	6e	67	3b	20	6e	6f	20	62	69	74	20	moving; no bit
060	6e	6f	72	20	6f	63	74	65	74	2e	0a	20	20	20	54	68	nor octet.. Th
070	65	20	65	6e	67	69	6e	65	65	72	73	20	72	61	74	74	e engineers ratt
080	6c	65	64	20	74	68	65	69	72	20	63	61	72	64	73	20	led their cards
090	69	6e	20	64	65	73	70	61	69	72	2c	0a	20	20	20	20	in despair,.
0a0	20	68	6f	70	69	6e	67	20	61	20	62	61	64	20	63	68	hoping a bad ch
0b0	69	70	20	77	6f	75	6c	64	20	62	6c	6f	77	20	77	69	ip would blow wi
0c0	74	68	20	61	20	66	6c	61	72	65	2e	0a	20	20	20	54	th a flare.. T
0d0	68	65	20	73	61	6c	65	73	6d	65	6e	20	77	65	72	65	he salesmen were
0e0	20	6e	65	73	74	6c	65	64	20	61	6c	6c	20	73	6e	75	nestled all snu
0f0	67	20	69	6e	20	74	68	65	69	72	20	62	65	64	73	2c	g in their beds,
100	0a	20	20	20	20	20	77	68	69	6c	65	20	76	69	73	69	. while visi
110	6f	6e	73	20	6f	66	20	64	61	74	61	20	6e	65	74	73	ons of data nets
120	20	64	61	6e	63	65	64	20	69	6e	20	74	68	65	69	72	danced in their
130	20	68	65	61	64	73	2e	0a	20	20	20	41	6e	64	20	49	heads.. And I
140	20	77	69	74	68	20	6d	79	20	64	61	74	61	73	63	6f	with my datasco
150	70	65	20	74	72	61	63	69	6e	67	73	20	61	6e	64	20	pe tracings and
160	64	75	6d	70	73	0a	20	20	20	20	20	70	72	65	70	61	dumps. prepa
170	72	65	64	20	66	6f	72	20	73	6f	6d	65	20	70	72	65	red for some pre
180	74	74	79	20	62	61	64	20	62	72	75	69	73	65	73	20	tty bad bruises
190	61	6e	64	20	6c	75	6d	70	73	2e	0a	20	20	20	57	68	and lumps.. Wh
1a0	65	6e	20	6f	75	74	20	69	6e	20	74	68	65	20	68	61	en out in the ha
1b0	6c	6c	20	74	68	65	72	65	20	61	72	6f	73	65	20	73	ll there arose s
1c0	75	63	68	20	61	20	63	6c	61	74	74	65	72	2c	0a	20	uch a clatter,.
1d0	20	20	20	20	49	20	73	70	72	61	6e	67	20	66	72	6f	I sprang fro
1e0	6d	20	6d	79	20	64	65	73	6b	20	74	6f	20	73	65	65	m my desk to see
1f0	20	77	68	61	74	20	77	61	73	20	74	68	65	20	6d	61	what was the ma

Chà, điều đó chắc chắn trông giống như phần mở đầu của một bài thơ! Mỗi dòng văn bản được phân tách bằng một ký tự đặc biệt gọi là *dòng mới*, có mã 0x0a (10 thập phân). Một số đầu tiên trong số này được đánh dấu bằng màu đỏ.

Chúng tôi đã gần hoàn thành. Rõ ràng là có nhiều tệp này hơn và để tìm thấy phần còn lại của nó, chúng tôi cần tham khảo FAT. Hãy nhớ lại rằng số *cụm* bắt đầu của tệp (khối chúng ta vừa xem) là 0xf4e. Mỗi mục nhập trong FAT có kích thước hai byte, vì vậy chúng tôi sẽ tìm mục nhập cho cụm đó ở độ lệch 0xf4e\*2 trong FAT, độ lệch này là 0x1e9c (việc cộng giá trị hai lần sẽ dễ dàng hơn so với thử nhân). Chúng tôi biết rằng một khối đĩa (và do đó một khối FAT) có kích thước 0x200 byte, vì vậy chúng tôi chỉ cần chia 0x1e9c cho 0x200. Điều này nghe có vẻ khó khăn nhưng không phải vậy. Bạn có thể tìm các công cụ cho việc này hoặc tự mình làm. Chúng ta hãy xem hai số này ở dạng nhị phân:

```
0x0200 => 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0x1e9c => 0 0 0 1 1 1 1 0 1 0 0 1 1 1 0 0
```

Số đầu tiên là lũy thừa của hai, vì vậy để chia cho nó, chúng ta chỉ cần dịch số thứ hai sang phải - trong trường hợp này là chín vị trí:

```
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 => 0x0f
```



Vì vậy, mục chúng tôi muốn nằm trong khối 0x0f của FAT. Tất nhiên, phần còn lại từ phép chia của chúng ta là tất cả những gì chúng ta đã mất khi dịch chuyển:

0 1 0 0 1 1 1 0 0 => 0x9c

vì vậy đây là độ lệch byte của mục trong khối FAT.

Chúng ta cần tìm khối FAT 0x0f. Chúng tôi biết FAT bắt đầu ở khối 1 của đĩa (xem trước đó), vì vậy khối 0x0f của FAT sẽ nằm trong khối đĩa 0x0f+1 hoặc khối 0x10. Hãy nhìn vào khối đó:

Block 16 (0x0010)																
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
090	00	00	00	00	00	00	00	00	00	00	00	00	4f	0f	50	0f
0a0	51	0f	ff	ff	00	00	00	00	00	00	00	00	00	00	00	00
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Chúng ta cần xem mục nhập FAT (hai byte) ở offset 0x9c; phần này được đánh dấu màu đỏ ở trên và phân giải thành giá trị 16 bit 0x0f4f. Đây thực sự là cụm tiếp theo, về mặt số lượng, từ cụm mà chúng ta vừa xem xét (điều này không phải lúc nào cũng đúng), vì vậy chúng ta có thể áp dụng một chút hiểu biết thông thường và suy ra rằng khối dữ liệu thứ hai của tệp xuất hiện ngay sau đó. cái đầu tiên; do đó, hai khối đầu tiên có kích thước 0xf65 và 0xf66. Đây là khối 0xf66:

```

Block 3942 (0x0f66)
  0 1 2 3 4 5 6 7 8 9 a b c d e f
000 74 74 65 72 2e 0a 0a 20 20 20 54 68 65 72 65 20 tter... There
010 73 74 6f 6f 64 20 61 74 20 74 68 65 20 74 68 72 stood at the thr
020 65 73 68 6f 6c 64 20 77 69 74 68 20 50 43 20 69 eshold with PC i
030 6e 20 74 6f 77 2c 0a 20 20 20 20 20 41 6e 20 41 n tow,. An A
040 52 50 41 4e 45 54 20 68 61 63 6b 65 72 2c 20 61 RPANET hacker, a
050 6c 6c 20 72 65 61 64 79 20 74 6f 20 67 6f 2e 0a ll ready to go..
060 20 20 20 49 20 63 6f 75 6c 64 20 73 65 65 20 66 I could see f
070 72 6f 6d 20 74 68 65 20 63 72 65 61 73 65 73 20 rom the creases
080 74 68 61 74 20 63 6f 76 65 72 65 64 20 68 69 73 that covered his
090 20 62 72 6f 77 2c 0a 20 20 20 20 20 68 65 27 64 brow,. he'd
0a0 20 63 6f 6e 71 75 65 72 20 74 68 65 20 63 72 69 conquer the cri
0b0 73 69 73 20 63 6f 6e 66 72 6f 6e 74 69 6e 67 20 sis confronting
0c0 68 69 6d 20 6e 6f 77 2e 0a 20 20 20 4d 6f 72 65 him now.. More
0d0 20 72 61 70 69 64 20 74 68 61 6e 20 65 61 67 6c rapid than eagl
0e0 65 73 2c 20 68 65 20 63 68 65 63 6b 65 64 20 65 es, he checked e
0f0 61 63 68 20 61 6c 61 72 6d 0a 20 20 20 20 20 61 ach alarm. a
100 6e 64 20 73 63 72 75 74 69 6e 69 7a 65 64 20 65 nd scrutinized e
110 61 63 68 20 66 6f 72 20 69 74 73 20 70 6f 74 65 ach for its pote
120 6e 74 69 61 6c 20 68 61 72 6d 2e 0a 0a 20 20 20 ntial harm...
130 4f 6e 20 4c 41 50 42 2c 20 6f 6e 20 4f 53 49 2c On LAPB, on OSI,
140 20 58 2e 32 35 21 0a 20 20 20 20 20 54 43 50 2c X.25!. TCP,
150 20 53 4e 41 2c 20 56 2e 33 35 21 0a 0a 20 20 20 SNA, V.35!..
160 48 69 73 20 65 79 65 73 20 77 65 72 65 20 61 66 His eyes were af
170 69 72 65 20 77 69 74 68 20 74 68 65 20 73 74 72 ire with the str
180 65 6e 67 74 68 20 6f 66 20 68 69 73 20 67 61 7a ength of his gaz
190 65 3b 0a 20 20 20 20 20 6e 6f 20 62 75 67 20 63 e;. no bug c
1a0 6f 75 6c 64 20 68 69 64 65 20 6c 6f 6e 67 3b 20 ould hide long;
1b0 6e 6f 74 20 66 6f 72 20 68 6f 75 72 73 20 6f 72 not for hours or
1c0 20 64 61 79 73 2e 0a 20 20 20 41 20 77 69 6e 6b days.. A wink
1d0 20 6f 66 20 68 69 73 20 65 79 65 20 61 6e 64 20 of his eye and
1e0 61 20 74 77 69 74 63 68 20 6f 66 20 68 69 73 20 a twitch of his
1f0 68 65 61 64 2c 0a 20 20 20 20 20 73 6f 6f 6e 20 head,. soon

```

mà chắc chắn trông giống như phần tiếp theo của bài thơ. Nếu chúng ta xem mục FAT cho cụm mới này (vì là khối tiếp theo nên cũng sẽ là cụm tiếp theo và do đó nằm trong mục FAT tiếp theo), nó được đánh dấu màu xanh lam ở trên và chứa giá trị 0x0f50. Đây là khối và cụm tiếp theo:

Block 3943 (0x0f67)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	67	61	76	65	20	6d	65	20	74	6f	20	6b	6e	6f	77	20	gave me to know
010	49	20	68	61	64	20	6c	69	74	74	6c	65	20	74	6f	20	I had little to
020	64	72	65	61	64	2e	0a	20	20	20	48	65	20	73	70	6f	dread.. He spo
030	6b	65	20	6e	6f	74	20	61	20	77	6f	72	64	2c	20	62	ke not a word, b
040	75	74	20	77	65	6e	74	20	73	74	72	61	69	67	68	74	ut went straight
050	20	74	6f	20	68	69	73	20	77	6f	72	6b	2c	0a	20	20	to his work,.
060	20	20	20	66	69	78	69	6e	67	20	61	20	6e	65	74	20	fixing a net
070	74	68	61	74	20	68	61	64	20	67	6f	6e	65	20	70	6c	that had gone pl
080	75	6d	62	20	62	65	72	73	65	72	6b	3b	0a	20	20	20	umb berserk;.
090	41	6e	64	20	6c	61	79	69	6e	67	20	61	20	66	69	6e	And laying a fin
0a0	67	65	72	20	6f	6e	20	6f	6e	65	20	73	75	73	70	65	ger on one suspe
0b0	63	74	20	6c	69	6e	65	2c	0a	20	20	20	20	20	68	65	ct line,. he
0c0	20	65	6e	74	65	72	65	64	20	61	20	70	61	74	63	68	entered a patch
0d0	20	61	6e	64	20	74	68	65	20	6e	65	74	20	63	61	6d	and the net cam
0e0	65	20	75	70	20	66	69	6e	65	21	0a	0a	20	20	20	54	e up fine!.. T
0f0	68	65	20	70	61	63	6b	65	74	73	20	66	6c	6f	77	65	he packets flowe
100	64	20	6e	65	61	74	6c	79	20	61	6e	64	20	70	72	6f	d neatly and pro
110	74	6f	63	6f	6c	73	20	6d	61	74	63	68	65	64	3b	0a	ocols matched;.
120	20	20	20	20	20	74	68	65	20	68	6f	73	74	73	20	69	the hosts i
130	6e	74	65	72	66	61	63	65	64	20	61	6e	64	20	73	68	nterfaced and sh
140	69	66	74	2d	72	65	67	69	73	74	65	72	73	20	6c	61	ift-registers la
150	74	63	68	65	64	2e	0a	20	20	20	48	65	20	74	65	73	tched.. He tes
160	74	65	64	20	74	68	65	20	73	79	73	74	65	6d	20	66	ted the system f
170	72	6f	6d	20	47	61	74	65	77	61	79	20	74	6f	20	50	rom Gateway to P
180	41	44	3b	0a	20	20	20	20	20	6e	6f	74	20	6f	6e	65	AD;. not one
190	20	62	69	74	20	77	61	73	20	64	72	6f	70	70	65	64	bit was dropped
1a0	3b	20	6e	6f	20	63	68	65	63	6b	73	75	6d	20	77	61	; no checksum wa
1b0	73	20	62	61	64	2e	0a	20	20	20	41	74	20	6c	61	73	s bad.. At las
1c0	74	20	68	65	20	77	61	73	20	66	69	6e	69	73	68	65	t he was finishe
1d0	64	20	61	6e	64	20	77	65	61	72	69	6c	79	20	73	69	d and wearily si
1e0	67	68	65	64	0a	20	20	20	20	20	61	6e	64	20	74	75	ghed. and tu
1f0	72	6e	65	64	20	74	6f	20	65	78	70	6c	61	69	6e	20	rned to explain

Chúng tôi tiếp tục điều này (một lần nữa, đó là khối và cụm tiếp theo) và chúng tôi tìm thấy 0x0f51 làm số cụm (được đánh dấu bằng màu xanh lục ở trên). Đây là khối đó:



Block 3944 (0x0f68)																	
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000	77	68	79	20	74	68	65	20	73	79	73	74	65	6d	20	68	why the system h
010	61	64	20	64	69	65	64	2e	0a	20	20	20	49	20	74	77	ad died.. I tw
020	69	73	74	65	64	20	6d	79	20	66	69	6e	67	65	72	73	isted my fingers
030	20	61	6e	64	20	63	6f	75	6e	74	65	64	20	74	6f	20	and counted to
040	74	65	6e	3b	0a	20	20	20	20	20	61	6e	20	6f	66	66	ten;. an off
050	2d	62	79	2d	6f	6e	65	20	69	6e	64	65	78	20	68	61	-by-one index ha
060	64	20	64	6f	6e	65	20	69	74	20	61	67	61	69	6e	2e	d done it again.
070	2e	2e	0a	0a	20	20	20	56	69	6e	74	20	43	65	72	66	.... Vint Cerf
080	0a	20	20	20	44	65	63	65	6d	62	65	72	20	31	39	38	. December 198
090	35	0a	00	00	00	00	00	00	00	00	00	00	00	00	00	00	5.....
0a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
1f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Cuối cùng, chúng ta xem mục nhập FAT cho khối/cụm này (được đánh dấu màu đen). Lần này mục nhập là 0xffff, cho biết rằng không còn khối nào trong tệp nữa. Chúng tôi đã hoàn thành!

## Người giới thiệu

Có một trang hữu ích tại <https://fromthegroundupmyway.blogspot.com/2020/10/it-is-time-to-look-into-fat.html>.

## Phản kết luận

Nếu bạn đã đạt được điều này (và hiểu tất cả) thì bạn đã hiểu rõ về hệ thống tệp FAT 16-bit. Bạn sẽ có thể phân tích một đĩa và xem liệu nó có bị hỏng hay không. Bạn thậm chí có thể sửa chữa nó!

