Chương 3:

HỆ THỐNG TẬP TIN

VÀ CÁC KIẾN TRÚC TỔ CHỨC PHỔ BIẾN

3.1. KIẾN TRÚC HỆ THỐNG TẬP TIN TỔNG QUÁT

3.1.1. KHÁI NIỆM

Các máy tính cao cấp thường có rất nhiều thông tin /dữ liệu, và việc truy xuất chúng diễn ra rất thường xuyên. Đa số thông tin trên máy tính cần phải được lưu giữ lâu dài, không mất khi không còn nguồn năng lượng cung cấp cho hệ thống lưu trữ. *(thông tin /dữ liệu ở đây được hiểu theo nghĩa tổng quát, bao gồm mọi thứ cần lưu trữ trên máy như hình ảnh, âm thanh, văn bản, tài liệu, trò chơi, chương trình ứng dụng, hệ điều hành, ...)*

Để thông tin lưu trữ được nhiều và không bị mất khi tắt máy thì chúng cần được lưu vào phần bộ nhớ thứ cấp (thường là hệ thống đĩa). Và khi khối lượng dữ liệu lưu trong máy quá lớn thì việc tổ chức, sắp xếp chúng sao cho có thể truy cập nhanh chóng hiệu quả sẽ trở nên rất quan trọng – nếu không khéo có thể khó quản lý được và thời gian truy cập sẽ rất rất lâu.

Việc lưu giữ thông tin, tổ chức /sắp xếp thông tin và cung cấp cho người sử dụng các thao tác cần thiết (chép, xóa, xem, sửa,...) không phải do máy tính hỗ trợ mà được thực hiện bởi hệ điều hành (HĐH), cụ thể là bởi thành phần hệ thống quản lý tập tin.

Tập Tin

Trên mỗi hệ thống bộ nhớ ngoài ta có thể lưu trữ được rất nhiều thông tin, vì dung lượng bộ nhớ ngoài thường rất lớn. Khi đó để có thể dễ dàng truy xuất & quản lý ta cần đặt cho mỗi thông tin một cái tên tương ứng, đồng thời khi lưu trữ nội dung thông tin ta cũng cần gắn kèm những thuộc tính cần thiết cho sự quản lý của người sử dụng hoặc của HĐH như kích thước của thông tin, ngày giờ tạo ra,... Tập hợp những thứ đó được gọi là tập tin.

☞ Tập tin là đơn vị lưu trữ thông tin trên bộ nhớ ngoài (hệ thống lưu trữ được lâu dài), mỗi tập tin bao gồm các thành phần: tên thông tin, nội dung của thông tin, kích thước của phần nội dung đó, và các thuộc tính cần thiết cho người sử dụng hoặc cho HĐH.

Tóm lại, khái niệm tập tin không phải có sẵn trên máy tính mà được đưa ra bởi HĐH (chính xác hơn là bởi những người làm ra phần mềm HĐH), nhằm giúp cho việc truy xuất & quản lý thông tin trên bộ nhớ ngoài được dễ dàng thuận lợi hơn.

Thư mục

Một khi số lượng tập tin được lưu trữ trên bộ nhớ ngoài đã lên tới một con số khá lớn thì nhất thiết phải đưa ra khái niệm thư mục. Khái niệm này cũng gần giống như hệ thống thư mục trong thư viện (khi đó mỗi tập tin có thể ví như một cuốn sách).

Nếu sách trong thư viện không được tổ chức theo một trật tự hợp lý mà cứ để chung vào một chỗ thì khi muốn tìm một cuốn sách, ta phải dò từng cuốn một cho đến khi tìm được. Cách làm đó có thể khiến ta phải mất rất nhiều thời gian, nhất là khi nó không có trong thư viện đó! Để khắc phục người ta đã tổ chức hệ thống thư mục: danh mục tên sách được liệt kê theo từng chủ đề, trong mỗi chủ đề lại có thể có nhiều chủ đề con. Việc tổ chức phân cấp dạng cây như vậy chắc chắn sẽ giúp cho thời gian tìm kiếm một cuốn sách theo một chủ đề nào đó sẽ rất nhanh – ngay cả khi chưa biết tên sách.

Số lượng tập tin trên bộ nhớ ngoài thường rất nhiều và nội dung của chúng còn đa dạng hơn nội dung của các quyển sách (không chỉ là tài liệu văn bản bình thường mà còn có thể là hình ảnh, âm thanh, trò chơi, chương trình...), do đó việc tìm kiếm & sử dụng chúng cũng sẽ còn khó khăn hơn nếu như ta không tổ chức theo một trật tự hợp lý. Vì vậy tổ chức hệ thống thư mục phân cấp trên bộ nhớ ngoài là rất cần thiết.

Tóm lại, khái niệm thư mục cũng được đưa ra bởi HĐH, để việc tìm kiếm & sử dụng tập tin được hiệu quả. Mỗi thư mục có thể chứa các tập tin và các thư mục con bên trong (dĩ nhiên

trong mỗi thư mục cũng có thể chỉ chứa toàn tập tin, hoặc chỉ chứa toàn thư mục con, hoặc đang là thư mục rỗng – không chứa gì cả).

Volume

Là một dãy "sector" được tổ chức theo một kiến trúc hợp lý để có thể lưu trữ lên đó một hệ thống tập tin & thư mục.

Trên máy tính có nhiều hệ thống lưu trữ, thông thường nếu hệ thống lưu trữ có kích thước không quá lớn (như thẻ nhớ, đĩa usb flash, CD, DVD, ...) thì sẽ được tổ chức thành một volume (vol). Với hệ thống lưu trữ lớn thì có thể tổ chức trên đó nhiều vol (như đĩa cứng), vì khi đó dung lượng vol nhỏ đi thì việc quản lý sẽ dễ dàng nhanh chóng hơn và độ an toàn dữ liệu cao hơn, đồng thời trên mỗi vol có thể tổ chức một kiến trúc lưu trữ khác nhau để phù hợp hơn cho các dạng dữ liệu được lưu trên đó.

Volume không nhất thiết phải thuộc một thiết bị lưu trữ vật lý, các "sector" trên volume không bắt buộc phải gắn kết với một vị trí vật lý cố định (tập tin nén chứa được một hệ thống thư mục và tập tin con bên trong có thể xem là một volume đặc biệt), các "sector" trên volume cũng có thể volume được nối kết từ nhiều thiết bị lưu trữ khác nhau.

3.1.2. THIẾT KẾ MÔ HÌNH

Mô hình thuộc tính tập tin

- Nội dung tập tin: Là dãy byte tuần tự, dãy các record chiều dài cố định hay theo cấu trúc cây; được lưu trữ dưới dạng nén hay không nén, nếu có nén thì dùng những phương pháp nén nào và cơ chế phân tích tập tin để xác định phương pháp nén tương ứng là như thế nào; …

- Tên tập tin: Ngoài phần tên chính ra có phần tên mở rộng hay không, bao nhiêu phần mở rộng; các thông tin chi tiết tương ứng cho từng phần như thế nào (*chiều dài tối đa & tối thiểu, sử dụng bảng mã nào, có phân biệt chữ cái thường /hoa không, có cho dùng khoảng trắng không, các ký*

tự không được dùng, các ký tự đại diện, các chuỗi không được trùng, giá trị các byte rác khi tên chưa đạt tới chiều dài tối đa (hoặc byte kết thúc/ chiều dài),…)

- Kích thước*:* Ngoài kích thước phần nội dung tập tin còn có kích thước phần nào khác không; lưu trữ bằng bao nhiêu byte, dưới dạng số nguyên không dấu hay có dấu;…

- Các thời điểm: ngày-tháng-năm & giờ-phút-giây lưu thành các trường riêng biệt hay ghép lại thành một dãy bit (thời điểm sử dụng gần nhất nếu có thì không cần thông tin giờ).

Ví dụ, nếu lưu theo hình thức: ***2B cho ngày*** *với ngày chiếm 5bit, tháng 4bit, năm 7bit (lưu giá trị của hiệu <năm> - 1980);* ***2B cho giờ*** *với giây chiếm 5bit (lưu giá trị <giây>/2), phút 6bit, giờ 5bit - thì thời điểm* ***12/3/2016 – 12:34:56*** *có dãy byte tương ứng là:* **52 35 5C 64** *(vì 3552h = 0100100001101100b, 645Ch = 0110010001011100b)*

- Mật khẩu: chuỗi tối đa và tối thiểu bao nhiêu ký tự, bao gồm và không bao gồm những ký tự nào,… (nếu khác rỗng thì sau khi kiểm tra đúng mới cho truy xuất nội dung tập tin)

- Các thuộc tính trạng thái: mỗi thuộc tính chỉ cần dùng 1 bit để biểu diễn nên tất cả các thuộc tính trạng thái nên ghép chung lại và lưu trữ bằng 1 byte hoặc 1 dãy byte vừa đủ (nếu số thuộc tính trạng thái > 8)

- …

Mô hình thuộc tính thư mục

Thiết kế tương tự như trên, và vì có khá nhiều thuộc tính giống với các thuộc tính của tập tin nên nhiều HĐH tổ chức một mô hình chung cho cả tập tin lẫn thư mục. Khi này thư mục được coi là một tập tin đặc biệt và có một thuộc tính để phân biệt với tập tin bình thường. Như vậy trong các thiết kế phục vụ cho việc lưu trữ & truy xuất, từ "tập tin" sẽ được hiểu là tập tin tổng quát – bao gồm tập tin bình thường và tập tin thư mục.

Mô hình chức năng

Tùy thuộc vào mục đích thiết kế và nhu cầu người sử dụng mà hệ thống sẽ có những chức năng cơ bản gì trên các tập tin & thư mục vừa xây dựng mô hình, ví dụ về một số chức năng cơ bản :

- Liệt kê danh sách các tập tin: ở gốc /ở thư mục con / cây thư mục, liệt kê theo thứ tự của thuộc tính nào, các thuộc tính nào cần xuất kèm, hình thức hiển thị như thế nào, …

- Đổi tên: phải kiểm tra người dùng hiện tại có được quyền thực hiện hay không, tên mới có hợp lệ không, có bị trùng với tập tin khác hay không.

- Đặt /đổi mật khẩu: phải hỏi mật khẩu cũ (nếu có), nếu đúng thì yêu cầu nhập mật khẩu mới và chỉ chấp nhận khi thỏa mãn đầy đủ các ràng buộc cần thiết (nội dung nhập ở 2 lần hỏi giống nhau, đáp ứng được qui định về chiều dài tối thiểu & tối đa, không phải là những chuỗi đặc biệt dễ đoán như trùng với tên tập tin /tên người dùng,…); nội dung tập tin phải được mã hóa tương ứng với mật khẩu; phải có cơ chế đề kháng với các hệ thống dò mật khẩu tự động;…

- Đặt /đổi thuộc tính: phải xác định thuộc tính tương ứng có được phép thay đổi hay không & tính hợp lệ của giá trị mới.

- Xóa tập tin: xóa bình thường /xóa nhưng không mất /xóa mất hẳn /xóa rác.

- Xóa phần mềm: xóa hết các tập tin của phần mềm (phải theo dõi và ghi nhận tên & vị trí khi cài đặt phần mềm), phục hồi lại các thay đổi mà phần mềm đã thiết đặt đối với hệ thống.

- Xóa thư mục: …

- …

3.1.3. TỔ CHỨC TRIỂN KHAI LÊN KHÔNG GIAN LƯU TRỮ

A. Các nhận xét & phân tích cần thiết:

Sau khi xây dựng xong mô hình tập tin, ta cần phải nghĩ đến hình thức lưu giữ chúng trên không gian lưu trữ. Ta phải thiết kế những kiến trúc hợp lý trên không gian lưu trữ để không chỉ có thể thực hiện được các thao tác cần thiết trên tập tin mà còn đạt được các hiệu ứng an toàn & tốc độ (đặc biệt là tốc độ đọc nội dung tập tin).

Không gian lưu giữ các tập tin được gọi là volume, là một dãy những sector (tổng quát hơn là block). Vì vậy thiết kế nêu trên cũng chính là thiết kế mô hình volume. Để có thể thiết kế một cách hiệu quả thì trước tiên cần phải xác định đầy đủ những mục đích quan trọng và phải đưa ra được các phân tích, nhận xét sau:

i. *Phải biết các vị trí còn trống (để có thể lưu dữ liệu vào vol)*
ii. *Mỗi sector chỉ thuộc tối đa một tập tin (để dễ quản lý, không bị nhầm lẫn thông tin giữa các tập tin)*
iii. *Tên và các thuộc tính của tập tin cần được lưu riêng vào một vùng (để tốc độ truy xuất dữ liệu nhanh hơn)*
iv. *Phải biết vị trí bắt đầu của nội dung tập tin (do phân tích trên)*
v. *Nội dung tập tin có bắt buộc phải liên tục hay không?*
vi. *Phải biết các vị trí chứa nội dung tập tin (nếu nội dung tập tin không bắt buộc liên tục)*
vii. *Phải biết các vị trí bị hư*
viii. *Nội dung tập tin nên lưu trữ theo đơn vị là CLUSTER (là dãy N sector liên tiếp – để dễ quản lý & việc truy xuất được nhanh hơn)*

B. Cluster

Nếu số sector trên vùng dữ liệu quá nhiều thì có thể sẽ khó hoặc không quản lý được, khi đó quản lý theo nhóm sector sẽ dễ dàng hiệu quả hơn. Ngoài ra nội dung tập tin thường chiếm nhiều sector và có thể không liên tục, lưu giữ nội dung theo từng nhóm sector sẽ làm giảm đi sự phân mảnh, tức dữ liệu sẽ an toàn hơn và tăng thời gian truy xuất nhanh hơn. Đồng thời ta còn

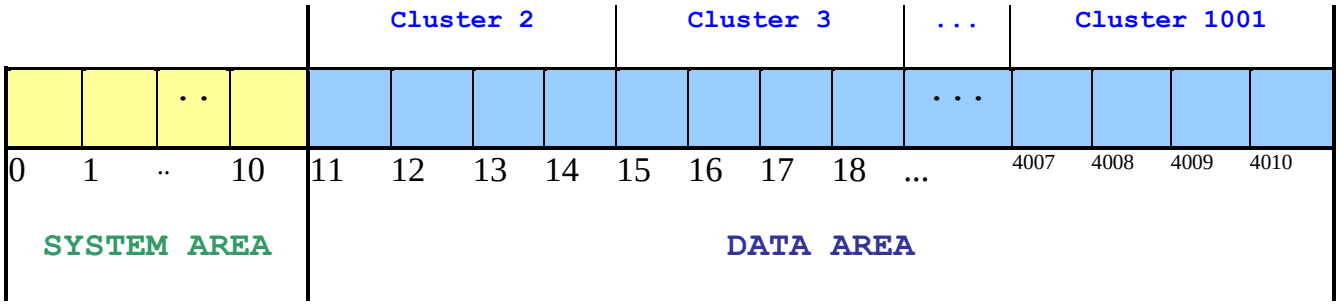
có thời gian đọc ghi một lần n sector liên tiếp thường nhanh hơn so với thời gian đọc ghi n lần mà mỗi lần chỉ 1 sector.

Như vậy, tuy đơn vị đọc ghi trên đĩa là sector nhưng đơn vị lưu trữ nội dung tập tin nên là cluster. Mỗi cluster là một dãy N sector liên tiếp. *(và mỗi vị trí trong các phân tích trên sẽ là một cluster)*

Thông thường Volume được chia thành 2 vùng: vùng dữ liệu (Data Area) chứa nội dung tập tin và vùng hệ thống (System Area) chứa các thông tin quản lý. Vùng System có kích thước nhỏ hơn nhiều so với vùng Data và phải truy xuất mỗi khi sử dụng Volume nên thường nằm trước. Cũng có những loại volume được thiết kế theo dạng không tách biệt hai vùng này (xem như vùng Data chiếm toàn bộ volume)

Vùng DATA là một dãy các cluster liên tiếp được đánh chỉ số theo thứ tự tăng dần (bắt đầu từ 0, 1 hay 2… tùy theo HĐH). Như vậy nếu vùng DATA có $\mathbf{S_D}$ sector & bắt đầu tại sector $\mathbf{S_S}$, mỗi cluster chiếm $\mathbf{S_C}$ sector, cluster đầu tiên có chỉ số là $\mathbf{F_C}$, thì *số cluster của vol* = $\mathbf{[S_D/Sc],}$ và *Cluster C sẽ bắt đầu tại sector*: $\mathbf{S_S+(C–F_C)*S_C}$

Ví dụ: nếu Vol X có kích thước 4014 sector, vùng SYSTEM chiếm 11 sector, mỗi cluster có 4 sector, cluster đầu tiên được đánh chỉ số là 2; thì phân bố cluster trên Vol sẽ như sau:



(3 sector 4011, 4012, 4013 sẽ không thuộc cluster nào và không được sử dụng)

Số sector trên một cluster nên là lũy thừa của 2 và có giá trị lớn hay nhỏ là tùy Vol. Nếu kích thước cluster càng lớn thì sẽ càng lãng phí không gian vì nội dung tập tin thường không chiếm trọn cluster cuối cùng, nhưng khi đó sẽ hạn chế được sự phân mảnh của tập tin và vì vậy tập tin có thể an toàn hơn & truy xuất nhanh hơn. Kích thước Cluster phụ thuộc vào nhiều yếu

tố: dung lượng vol, tốc độ truy xuất một dãy sector trên vol, kích thước của đa số tập tin sẽ lưu vào vol, số cluster tối đa mà hệ thống có thể quản lý, nhu cầu của người dùng,.. Trên các đĩa cứng hiện tại thì cluster thường là 4, 8 hoặc 16 sector. Với hệ thống chỉ có thể quản lý tối đa M cluster, nếu vùng Data có M*N sector thì phải cho mỗi cluster tối thiểu N sector mới có thể quản lý được toàn bộ volume.

C. Bảng quản lý Cluster

Để đưa nội dung tập tin vào vol thì phải xác định các cluster trống, để đọc nội dung tập tin ra thì phải xác định danh sách các cluster chứa nội dung tập tin đó. Và nếu muốn biết cluster đang ở trạng thái như thế nào mà phải truy xuất tới cluster mới biết (tức phải lưu thông tin quản lý ngay trên cluster) thì hệ thống sẽ có thể chạy rất chậm – vì số lượng cluster có thể rất lớn và thời gian truy xuất một cluster cũng hơi lâu. Vì vậy phải lập ra một hoặc một số bảng quản lý cluster để hệ thống truy xuất được nhanh chóng.

Về tổ chức thiết kế thì có thể dùng các cấu trúc dữ liệu khác nhau để quản lý các cluster - nhưng khi lưu chúng vào vol thì sẽ gọi chung là bảng (table). Ta có thể dùng một bảng chung, cũng có thể tổ chức thành nhiều bảng với mỗi bảng phục vụ cho một nhu cầu.

C.1. Thiết kế quản lý cluster hư:

Vấn đề này không phải quan trọng lắm, vì nhiều vol không có cluster hư vật lý. Tuy nhiên với các vol có khả năng phát sinh cluster hư thì cũng nên quản lý, có những cách thiết kế sau: l*ưu bằng một danh sách trực tiếp: giá trị mỗi phần tử là chỉ số của một cluster hư, vì số cluster hư rất ít nên danh sách này có thể qui định là một hoặc vài sector ; kết hợp với các trạng thái luận lý khác trong một bảng chỉ mục.*

C.2. Thiết kế quản lý cluster trống:

Để chép tập tin vào volume thì ta cần phải chia nội dung tập tin thành từng đoạn, mỗi đoạn có kích thước một cluster, và lưu mỗi đoạn vào một cluster trống. Vì vậy ta cần phải biết danh sách các cluster trống trên vol, và thậm chí trong danh sách các cluster trống của vol ta cần

phải chọn ra những cluster "hợp lý" nhất để lưu giữ nội dung tập tin (chẳng hạn đó là dãy cluster liên tiếp thì tốt hơn là không liên tiếp).

Ngoài mục đích lưu giữ nội dung tập tin, ta cũng cần biết tổng số cluster trống để biết dung lượng còn trống của vol, vị trí & kích thước của các vùng trống để có thể điều chỉnh (dồn - defragment) lại cho hệ thống hiệu quả hơn. Tuy có thể tổ chức thông tin trạng thái ngay trên cluster nhưng như vậy thì việc xác định danh sách các cluster trống sẽ rất chậm, vì đầu đọc phải thực hiện quá nhiều thao tác di chuyển /đọc. Nhất thiết phải có một cấu trúc dữ liệu riêng để quản lý các cluster trống, có những thiết kế như sau:

- *Dùng hình thức bitmap*: sử dụng một dãy byte, trong đó mỗi bit trên dãy quản lý một cluster. Nếu muốn biết cluster K là trống hay không ta xem giá trị của bit K là 0 hay 1. Dãy bit quản lý N cluster sẽ có kích thước [(N+7) div 8] byte, bit K trong dãy là bit (K mod 8) của byte (K div 8).
- *Quản lý theo dạng chỉ mục*: Để xác định các cluster trống, hư, hay thuộc trạng thái luận lý đặc biệt nào đó ta có thể quản lý theo dạng chỉ mục: mỗi phần tử của bảng quản lý là 1 con số nói lên trạng thái của cluster mang chỉ số tương ứng.
- *Quản lý vùng trống*: tổ chức một danh sách các phần tử, mỗi phần tử chứa vị trí bắt đầu & kích thước của vùng trống tương ứng.

C.3. Quản lý chuỗi các cluster chứa nội dung của tập tin:

Việc xác định một trạng thái luận lý của cluster có thể thiết kế khá đơn giản, vấn đề chủ yếu là hình thức tổ chức để có thể xác định chuỗi các cluster chứa nội dung của một tập tin, có một số hình thức cơ bản như sau:

- Lưu trữ nội dung tập tin trên dãy cluster liên tục (danh sách đặc).
- Sử dụng cấu trúc danh sách liên kết.
- Sử dụng cấu trúc cây.
- Sử dụng cấu trúc danh sách liên kết kết hợp chỉ mục (index).
- Sử dụng cấu trúc cây (kết hợp chỉ mục).

C.3.1 Lưu trữ nội dung tập tin trên dãy cluster liên tiếp*

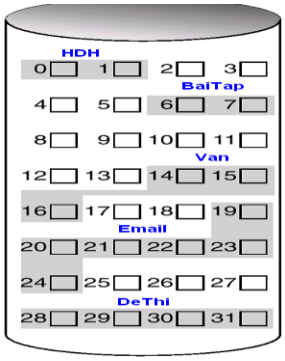
Nếu nội dung tập tin bắt buộc phải liên tục (trái với phân tích (v)) thì việc xác định các cluster chứa nội dung tập tin rất đơn giản: ta chỉ cần biết vị trí cluster bắt đầu và kích thước tập tin.

☞ Phương pháp này có những bất tiện lớn là tập tin khó tăng kích thước và có thể không lưu được một tập tin kích thước bình thường dù không gian trống còn rất lớn, vì có thể có rất nhiều vùng trống trên đĩa nhưng không có vùng nào đủ để chứa.

Ví dụ: Nếu cluster có kích thước là 1K, và các tập tin trên vol là:

- HDH: (0, 2006)

- BaiTap: (6, 2007)



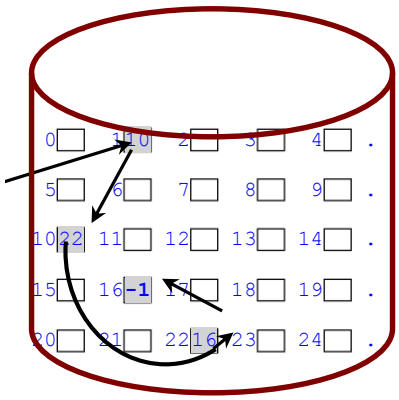
C.3.2 Sử dụng cấu trúc danh sách liên kết*

Mỗi cluster sẽ là một phần tử của danh sách liên kết, gồm hai vùng: vùng chứa nội dung tập tin và vùng liên kết chứa chỉ số cluster kế tiếp.

☞ Phương pháp này có một khuyết điểm lớn là việc xác định danh sách các cluster của tập tin sẽ rất chậm (vì phải truy xuất luôn nội dung tập tin), mà việc này lại phải thực hiện rất thường xuyên. Ta cần phải lấy danh sách các cluster chứa nội dung tập tin để sắp xếp thứ tự truy xuất cho hợp lý rồi mới truy xuất, để tối ưu thời gian truy xuất. (Ngoài ra để biết tập tin có bị phân mảnh không thì cũng phải xác định danh sách cluster của nó)

Ví dụ:

*Nếu tập tin chiếm các cluster là **1, 10, 22, 16** thì phân bố các cluster của những tập tin đó trong đĩa sẽ như hình*



*C.3.3 * Dùng cấu trúc danh sách liên kết kết hợp chỉ mục*

Ta tổ chức một bảng các phần tử nguyên (dãy số nguyên), mỗi phần tử được dùng để quản lý một cluster trên vùng dữ liệu theo dạng chỉ mục (**phần tử K quản lý cluster K**). Với qui định:

- Nếu phần tử K trên bảng có giá trị là FREE thì cluster K trên vùng dữ liệu đang ở trạng thái trống.
- Nếu phần tử K trên bảng có giá trị là BAD thì cluster K trên vùng dữ liệu sẽ được hệ thống hiểu là ở trạng thái hư.
- Nếu phần tử K trên bảng có giá trị khác FREE và khác BAD thì cluster K trên vùng dữ liệu đang chứa nội dung của 1 tập tin. Khi đó giá trị này cũng chính là chỉ số của cluster kế tiếp chứa nội dung của tập tin, nhưng nếu giá trị nàylà EOF thì không có cluster kế (cluster K đã là cluster cuối cùng của tập tin).

☞ Hình thức tổ chức này có thể đáp ứng được tất cả các nhu cầu quản lý cluster: xác định cluster trống, hư, hay đang chứa nội dung tập tin, và chuỗi các cluster chứa nội dung của tập tin (khi biết cluster bắt đầu).

Ví dụ, nếu nội dung tập tin chiếm các cluster là 1, 10, 22, 16 thì nội dung bảng như sau:

	10									**22**		
0	1	2	3	4	5	6	7	8	9	10	11	12

			Eof						**16**			
13	14	15	16	17	18	19	20	21	22	23	24	...

Với Volume X, nếu nội dung bảng quản lý Cluster như sau:

		3	**5**	**bad**	**eof**	**eof**	**11**	**free**	**7**	**bad**	**12**	**eof**	**free**	**...**	**free**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...	1001

(các phần tử từ 13 đến 1001 đều có giá trị là FREE)

thì từ đây có thể xác định lúc này trên volume đang có 2 cluster hư, 990 cluster trống, 8 cluster chứa nội dung tập tin. Cụ thể là:

- *Các cluster hư: 4, 10*
- *Các cluster trống: 8, 13, 14, 15, ..., 1001*
- *Các cluster chứa nội dung tập tin: 2, 3, 5, 6, 7, 9, 11, 12. Trong đó có 3 tập tin:*
  + tập tin I chiếm 3 cluster theo đúng thứ tự là: 2, 3, 5.
  + tập tin II chiếm 1 cluster duy nhất là: 6.
  + tập tin III chiếm 4 cluster theo đúng thứ tự là: 9, 7, 11, 12.

(Nếu biết cluster đầu của tập tin thì ta dễ dàng suy ra được các cluster kế tiếp, nhưng thiết kế này còn

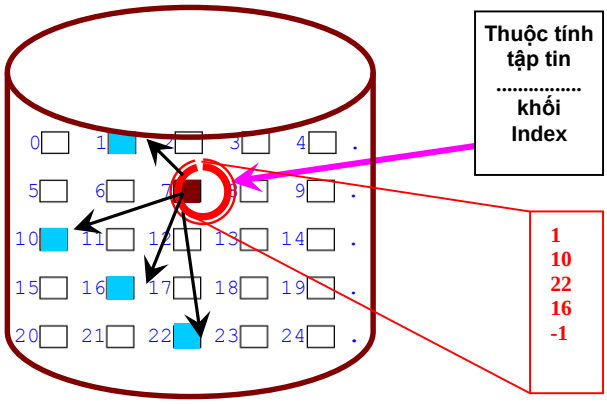
cho phép lần ngược)

Lưu ý: Phần tử đầu tiên của bảng có chỉ số là 0 nên nếu cluster đầu tiên của vùng DATA được đánh chỉ số là $F_C > 0$ thì F_C phần tử đầu tiên của bảng (từ 0.. F_C -1) sẽ không được dùng để quản lý cluster, để đảm bảo tính chất chỉ mục (phần tử mang chỉ số K quản lý cluster mang chỉ số K).

*C.3.4 * Sử dụng cấu trúc cây (kết hợp chỉ mục)*

Nếu số khối mà tập tin chiếm quá lớn thì quản lý bằng danh sách liên kết sẽ dễ gây ra hiện tượng mất mát dữ liệu nghiêm trọng, vì có quá nhiều con trỏ nên khả năng bị lỗi trên một con trỏ sẽ cao – và khi này tất cả các cluster từ vị trí đó đều sẽ bị thất lạc.

Vì vậy với những hệ thống lưu trữ siêu lớn ta cần phải dùng hình thức quản lý là cây nhiều nhánh, để nếu có lỗi trên một con trỏ thì chỉ mất một nhánh. Đồng thời cũng phải hạn chế tối đa số lượng con trỏ, bằng cách tại mỗi nút lá của cây ta lưu một số lượng khá lớn các chỉ số trực tiếp của các cluster chứa nội dung tập tin. Mỗi nút lá ở đây sẽ là một khối chỉ mục. Ví dụ, nếu tập tin chiếm các cluster là 1, 10, 22, 16 thì sơ đồ lưu trữ có thể như sau:

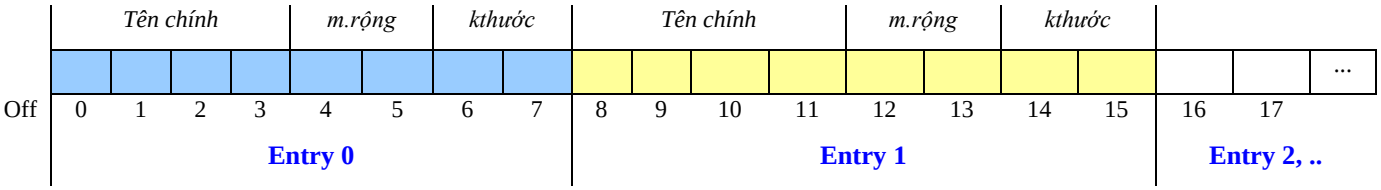


D. Bảng Thư Mục

Là một dãy phần tử (entry), mỗi phần tử chứa tên & các thuộc tính của một tập tin. Thường gồm 2 loại: bảng thư mục gốc (RDET - Root Directory Entry Table) và bảng thư mục con (SDET - Sub Directory Entry Table).

Cấu trúc của entry được tạo dựng từ mô hình thuộc tính của tập tin, có thể bổ sung một số byte dành riêng để khi đưa thêm những thuộc tính mới trong các version sau của HĐH thì hệ thống vẫn có khả năng tương thích (ngược).

Ví dụ, dãy entry của một bảng thư mục có thể như sau::



khi này tập tin chỉ gồm các thành phần: tên chính (tối đa 4 byte), tên mở rộng (tối đa 2 byte), kích thước (số nguyên 2 byte).

Bảng SDET chắc chắn sẽ nằm ở vùng DATA và không bị cố định về kích thước cũng như vị trí, vì thực chất SDET là một tập tin. Với RDET thì một số HĐH qui định phải nằm cố định trên vùng SYSTEM (như kiến trúc FAT12/FAT16). Nếu hệ thống không qui định RDET cố định thì có thể lưu trên vùng DATA như SDET, và quản lý nội dung như tập tin bình thường.

Bảng SDET ban đầu sẽ chiếm 1 cluster, sau khi đưa vào các tập tin thì tới một lúc nào đó có thể sẽ không còn entry trống để cấp. Khi này hệ thống sẽ tìm một cluster trống khác và nối thêm cho SDET. Điều này cứ tiếp tục xảy ra cho đến khi không còn cluster trống nữa, hoặc thư mục đã đạt tới kích thước giới hạn (nếu có). Thông thường kích thước của SDET là không giới hạn.

E. Boot Sector

Là sector đầu tiên của vùng SYSTEM, cũng như của vol. Boot Sector chứa một đoạn chương trình nhỏ để nạp HĐH khi khởi động máy (vì vậy mà có tên là "boot") và các thông số quan trọng của vol: kích thước vol, kích thước cluster, kích thước bảng quản lý cluster, ...

Các bảng quản lý cluster & bảng thư mục đã đủ đáp ứng các yêu cầu cần thiết, nhưng để thật sự có thể thực hiện được các thao tác truy xuất vol ta cần phải biết được vị trí & kích thước của từng thành phần trên vol. Vì vol có thể được kết nối vào một hệ thống khác nên thông tin về các thành phần của vol phải được lưu ngay trên chính vol đó để bất cứ hệ thống nào cũng có thể hiểu. Sector đầu tiên của vol là nơi thích hợp nhất để chứa các thông tin quan trọng này.

Các thông số quan trọng chỉ chiếm một kích thước nhỏ nên ta có thể qui ước một vùng nhỏ trên sector dùng để chứa các thông số của vol (phần còn lại là đoạn chương trình nạp HĐH khi khởi động). Mỗi thông số được qui định nằm tại một offset cụ thể cố định nào đó (với kích thước lưu trữ & kiểu dữ liệu tương ứng) trên sector 0. Tuy nhiên vẫn có thể lưu ở những sector kế tiếp - trong trường hợp tổng kích thước các thông số và phần chương trình nạp HĐH lớn hơn 512byte chẳng hạn.

Trên thực tế nhiều hệ thống có cả một vùng Boot Sector, bao gồm Boot sector và một số sector dự trữ /dành riêng, trong vùng này có thể có một bản backup của Boot sector và vài sector chứa các thông tin phụ. Với nhiều volume không dùng để khởi động thì đoạn Boot Code không có giá trị và nếu ta có ghi nội dung bất kỳ vào vùng BootCode này thì cũng không gây ra ảnh hưởng đến volume.

3.2. KIẾN TRÚC HỆ THỐNG TẬP TIN FAT

3.2.1. TỔNG QUAN

Kiến trúc hệ thống tập tin FAT (File Allocation Table) được giới thiệu từ năm 1977, là một kiến trúc có thiết kế đơn giản nhưng rất hiệu quả và được sử dụng cực kỳ rộng rãi trên rất nhiều hệ thống cho đến mãi những năm gần đây. Với các volume có dung lượng nhỏ (vài chục GiB đổ lại) chứa các file không quá lớn (dưới 4GiB) thì FAT vẫn tỏ ra rất ưu việt, còn khi kích thước file trên 4GiB hoặc volume lớn hơn 2TiB thì định dạng FAT không còn quản lý được và phải sử dụng hệ thống tập tin theo định dạng khác.

Thiết kế quan trọng nhất trong kiến trúc FAT là thiết kế cho bảng quản lý cluster (còn gọi là bảng FAT). Bảng này được tổ chức theo cấu trúc danh sách liên kết kết hợp chỉ mục. Thời gian đầu (từ 1977 đến 1988), kích thước volume rất nhỏ mà không gian lưu trữ lại rất quí giá, bảng FAT có thiết kế mỗi phần tử chỉ chiếm 12bit để không bị lãng phí dung lượng – và gọi là FAT12. Sau đó, khi bắt đầu phải quản lý tới các volume có dung lượng trên 32MiB, FAT16 (mỗi phần tử của bảng FAT chiếm 16bit) ra đời. Năm 1996, đến lượt FAT32 được giới thiệu để có thể sử dụng cho các volume trên 2GiB /4GiB. Thông thường, khi nói định dạng FAT thì nghĩa là

FAT12 hoặc FAT16, và những năm gần đây hầu như đã không còn dùng FAT12 nên FAT thường đồng nghĩa là FAT16 (và trên thực tế FAT16 cũng không còn dùng nhiều vì chỉ quản lý được volume tối đa 2GiB). Tuy nhiên vì các định dạng FAT12, FAT16, FAT32 khá tương đồng nên trong tài liệu này thì cả 3 định dạng đó có thể gọi chung là FAT.

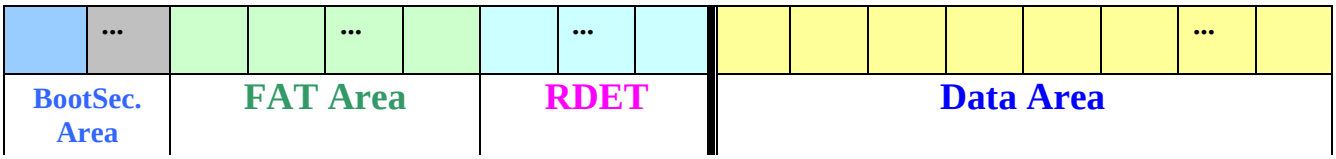
Các thiết kế FAT12, FAT16, FAT32 khá giống nhau - hầu như chỉ khác về kích thước phần tử và đều gặp các hạn chế về dung lượng (mặc dù dùng tới 32bit cho phần tử như FAT32 thì vẫn bị giới hạn khi kích thước file trên 4GiB hoặc volume lớn hơn 2TiB). Để khắc phục công ty Microsoft đã đưa ra thiết kế Extended FAT (exFAT), kiến trúc này hiệu quả hơn khi volume hoặc file có dung lượng lớn nhưng cũng có nhiều khác biệt và phức tạp hơn so với các kiến trúc FAT trước và sẽ được xét riêng.

3.2.2. ĐỊNH DẠNG FAT12 VÀ FAT16

Tổ chức Volume trên định dạng FAT12 và FAT16 bao gồm hai thành phần chính:

- Vùng dữ liệu (Data) chứa nội dung tập tin, được tổ chức dưới đơn vị cluster gồm nhiều sector để hệ thống hoạt động tối ưu, nằm phía sau.
- Vùng hệ thống (System) chứa các thông tin quản lý, bao gồm Boot Sector chứa các thông số quan trọng của volume, bảng FAT dùng để quản lý các cluster trên, bảng thư mục gốc (RDET) quản lý tên tập tin & các thuộc tính liên quan.

Các thành phần này có thứ tự phân bố như sau:



A. Vùng Data

Là vùng lưu trữ nội dung của các tập tin và thư mục, các sector trên vùng này được nhóm lại thành những cluster, mỗi cluster là một dãy các sector liên tiếp. Đơn vị truy xuất của phần cứng là sector nhưng đơn vị lưu trữ nội dung tập tin là cluster.

Cluster đầu tiên trên vùng Data được đánh chỉ số là 2. Nếu kích thước vùng System là **Ss** và kích thước cluster là **Sc** sector thì cluster C sẽ bắt đầu tại sector: **Ss+(C–2)*Sc**

B. Bảng FAT

Do bảng này rất quan trọng và thường xuyên được truy xuất nên thường được lưu thành nhiều bảng (N_F bảng) để phòng tránh hư hỏng, các bảng này nằm kế tiếp nhau trên một vùng gọi là vùng FAT. Số lượng bảng FAT của volume được lưu trữ bằng số nguyên 1 byte tại offset 10h của Boot Sector (thường $N_F = 2$).

Vị trí bắt đầu của vùng FAT cũng chính là kích thước vùng BootSector (viết tắt là S_B), vì vùng FAT nằm kế sau vùng BootSector. Thông số này được lưu trữ bằng số nguyên 2 byte tại offset Eh của Boot Sector.

Số sector của mỗi bảng FAT (S_F) được tổ chức bằng số nguyên 2 byte tại offset 16h của Boot Sector.

Kích thước vùng FAT là $\mathbf{N_F * S_F}$ (sector), nhưng kích thước thực sự sử dụng của bảng FAT có thể nhỏ hơn không gian dành cho nó, ví dụ bảng FAT chỉ cần 6.7 sector thì S_F được lưu tại offset 16h không nhất thiết phải là 7 mà có thể là 8.

Do phần tử đầu của bảng phải mang chỉ số là 0 mà cluster đầu tiên trên vùng Data có chỉ số là 2, nên 2 phần tử đầu của FAT không tương ứng với cluster nào. Lý do phải dùng FAT12 với kích thước phần tử lẻ (không phải là bội của 1 byte – 8bit) là vì không gian lưu trữ ngày xưa rất hạn chế nên cần phải cố gắng tiết kiệm. Hiện nay điều này không còn đúng nữa nên FAT12 không còn được sử dụng, và thậm chí FAT16 cũng ít được dùng – chủ yếu dùng các cải tiến mở rộng như FAT32 và exFAT

Nếu phần tử K của FAT có giá trị L thì trạng thái của Cluster K là:

Trạng thái của Cluster K	**Giá trị của L**		**Ghi chú**
	FAT12	**FAT16**	
Trống	*0*	**0**	**FREE**
Hư	*FF7*	**FFF7**	**BAD**
Là Cluster cuối của file	*FFF*	**FFFF**	**EOF**
Chứa nội dung file, và cluster kế sau là L	*2 .. FEF*	**2 .. FFEF**	**USED**

Có một số giá trị tuy vẫn thuộc phạm vi biểu diễn nhưng không thể có với L *(ví dụ, với FAT12 là các giá trị 1, FF0 .. FF6, FF8 .. FFE)*, đây là các giá trị dành riêng được dự phòng cho những phiên bản sau của HĐH, để có được sự tương thích giữa các phiên bản cũ & mới. Vì giá trị 0 được dùng để biểu diễn trạng thái cluster trống nên sẽ không thể có cluster mang chỉ số 0 trên vùng dữ liệu. Căn cứ vào những giá trị tương ứng với trạng thái cluster chứa nội dung tập tin, ta có thể nói chỉ số cluster lớn nhất sẽ không thể vượt quá FEFh hoặc FFEFh.

Cũng từ đó, ta có thể suy ra số cluster tối đa mà bảng FAT12 có thể quản lý được là FEEh (tức 4078d, chứ không phải là 2^{12} = 4096d), và FAT16 là FFEEh (tức 65518d). Như vậy nếu **số cluster không quá 4078 thì hệ thống sẽ dùng FAT12** để quản lý, nếu số cluster **lớn hơn 4078 nhưng không quá 65518 thì sẽ dùng FAT16**, và **lớn hơn 65518 thì sẽ dùng FAT32**. Tuy nhiên đó là qui ước mặc định của HĐH**, ta có thể chỉ định loại FAT**, ví dụ vùng DATA có 2006 cluster thì dùng FAT16 quản lý vẫn được.

Với FAT12, việc truy xuất một phần tử hơi phức tạp vì đơn vị truy xuất trên RAM là 1 byte nhưng mỗi phần tử lại có kích thước 1.5 byte. Ta có thể xác định 2 byte tương ứng chứa giá trị của phần tử, lấy giá trị số nguyên không dấu 2byte tại đó rồi dùng các phép toán xử lý trên bit để truy xuất được con số 1.5 byte tương ứng.

Ví dụ, nội dung 12 byte đầu của bảng FAT là:

	F0	**FF**	**FF**	**03**	**40**	**00**	**FF**	**7F**	**FF**	**AB**	**CD**	**EF**
Offset	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*

thì 8 phần tử đầu tương ứng của bảng FAT là:

	FF0	**FFF**	**003**	**004**	**FFF**	**FF7**	**DAB**	**EFC**
phầntử	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*

Khi lập trình truy xuất bảng FAT12, vì kích thước bảng là khá nhỏ (tối đa 12 sector) nên để việc truy xuất một phần tử được đơn giản ta cần đọc toàn bộ các sector trên bảng FAT12 vào một vùng đệm rồi từ vùng đệm này xây dựng ra một mảng số nguyên 16bit mà mỗi phần tử của mảng mang giá trị của phần tử tương ứng trên bảng FAT. Khi cần lưu bảng FAT vào vol thì thực hiện

thao tác ngược lại: từ mảng nguyên 16bit tạo ra dãy byte tương ứng của FAT đưa vào vùng đệm và lưu vùng đệm vào các sector chứa FAT trên vol.

C. Bảng RDET (Root Directory Entry Table)

Là một dãy entry liên tiếp, mỗi entry chứa tên và các thuộc tính của một tập tin /thư mục ở thư mục gốc (hoặc đang ở trạng thái trống). Các entry của những tập tin /thư mục không phải ở gốc được lưu trong các bảng SDET trên vùng Data.

C.1. Vị trí và kích thước

Bảng RDET trên kiến trúc FAT12 & FAT16 luôn nằm ở cuối vùng System, kế sau vùng FAT và kế trước vùng Data. Tức vị trí bắt đầu của RDET là $S_B + N_F * S_F$. Số entry của bảng RDET (S_R) được lưu trữ bằng số nguyên 2 byte tại offset 11h của Boot Sector. Mỗi entry trên RDET có 32 byte, chứa tên và các thuộc tính của tập tin. Khi mới định dạng volume, các entry sẽ đều ở trạng thái trống (32 byte trên entry đều là 0).

C.2. Cấu trúc Entry

C.2.1 Cấu trúc entry thư mục trên MSDOS*

Ban đầu (giai đoạn máy PC mới ra đời – năm 1980), không gian đĩa là rất quí và hệ điều hành MSDOS qui định tên tập tin phải ở dạng 8.3, tức phần tên chính của tập tin chỉ được phép dài tối đa là 8 ký tự - và phần tên mở rộng tối đa 3 ký tự.

Cấu trúc cụ thể của các entry thư mục trên DOS như sau:

Offset	Số byte	Kiểu lưu trữ	Ý nghĩa
0h (0)	8	Chuỗi ký tự ASCII	Tên chính của tập tin
8h (8)	3	Chuỗi ký tự ASCII	Tên mở rộng
Bh (11)	1	Số nguyên không dấu	Thuộc tính trạng thái (0.0.A.D.V.S.H.R)
Ch (12)	10	Chuỗi byte	Không dùng
16h (22)	2	Số nguyên không dấu	Giờ cập nhật tập tin (ss/2:5b; mm:6b; hh: 5b)

18h (24)	2	Số nguyên không dấu	Ngày cập nhật tập tin (dd:5b; mm:4b; yy-1980:7b)
1Ah (26)	2	Số nguyên không dấu	Cluster bắt đầu
1Ch (28)	4	Số nguyên không dấu	Kích thước tập tin (theo byte)

Hay:

	Tên			*Tttt*	*Ko dùng*			*Giờ*		*Ngày*		*Cluster bắtđầu*		*Kích thước*		
	X	X	X	Y	Z	Z	Z	G	G	N	N	C	C	K	K	K
Off	0	..	10	11	12	..	21	22	23	24	25	26	27	28	..	31

Trong đó:

- Tên chính của tập tin: được lưu bằng đúng 8 byte từ offset 0 đến offset 7 theo bảng mã ASCII, nếu tên tập tin không đủ 8 ký tự thì các byte còn lại sẽ lưu khoảng trắng (giá trị 20h), các ký tự chữ cái thường trên tên tập tin được đổi thành chữ cái hoa tương ứng.

- Tên mở rộng: được lưu bằng 3 byte từ offset 8 đến offset 10, nếu không đủ 3 ký tự thì các byte còn lại sẽ lưu khoảng trắng.

- Thuộc tính trạng thái: được lưu bằng 1 byte tại offset 11. Chứa 6 thuộc tính luận lý, mỗi thuộc tính tương ứng với một bit theo qui ước bit bật (1) là thuộc tính ở trạng thái có và bit tắt (0) là trạng thái không. Hai bit cao nhất không sử dụng và được qui định phải là 0. Các thuộc tính tương ứng với các bit trong byte như sau:

0	0	A	D	V	S	H	R
7	6	5	4	3	2	1	0

+ R (ReadOnly): thuộc tính chỉ đọc, khi có thuộc tính này hệ thống sẽ không cho phép sửa hoặc xóa.

+ H (Hidden): thuộc tính ẩn, ở trạng thái mặc định hệ thống sẽ không hiển thị tên của các tập tin này khi liệt kê danh sách tập tin.

+ S (System): hệ thống, cho biết tập tin có phải thuộc HĐH không.

+ V (VolumeLabel): nhãn vol, trên RDET chỉ có tối đa một entry có thuộc tính này, khi đó entry không phải tương ứng với tập tin mà được dùng để chứa nhãn của vol – là một chuỗi tối đa 11 ký tự được lưu ở đầu entry.

+ D (Directory): thuộc tính thư mục, nếu entry có thuộc tính này thì tập tin tương ứng không phải là một tập tin bình thường mà là một tập tin thư mục. Thư mục trên DOS được lưu trữ như một tập tin bình thường, nội dung của tập tin thư mục là danh mục các tập tin và thư mục con của nó.

+ A (Archive): thuộc tính lưu trữ, cho biết tập tin đã được backup hay chưa (bằng lệnh backup của HĐH), đây là thuộc tính hầu như không còn sử dụng – vì ít khi có nhu cầu liên tục backup tất cả các tập tin trên

vol.

- Giờ cập nhật cuối cùng: bao gồm giờ, phút và giây. Ba thông tin này được lưu trong 2 byte theo hình thức: giá trị giây/2 lưu trong 5 bit đầu tiên, phút chiếm 6 bit kế tiếp và 5 bit còn lại chứa giờ.

- Ngày cập nhật cuối cùng: tương tự 3 thông tin ngày-tháng-năm được lưu trong 2 byte, theo hình thức: ngày lưu bằng 5 bit đầu tiên, tháng chiếm 4 bit kế tiếp và 7 bit còn lại chứa giá trị <năm–1980>.

- Cluster bắt đầu: là số nguyên không dấu 2 byte dùng để lưu chỉ số cluster bắt đầu của nội dung tập tin (ví dụ nếu tập tin bắt đầu ngay tại đầu vùng dữ liệu thì giá trị của trường này sẽ là 2).

- Kích thước tập tin: là số nguyên không dấu 4 byte, lưu kích thước của phần nội dung tập tin (sẽ có giá trị 0 nếu entry là thư mục).

Ví dụ, nếu entry có nội dung như sau:

```
44 45 54 48 49 20 20 20 44 4F 43 20 00 00 00 00
00 00 00 00 00 00 16 4A BB 34 14 00 1E 0A 00 00
```

*thì tập tin tương ứng có tên là **Dethi.doc**, và các thuộc tính còn lại là:*

- *Kích thước = <số 4byte tại offset 1Ch> = A1Eh = 2590 byte*
- *Cluster bắt đầu = <số 4byte tại offset 1Ah> = 14h = 20*
- *Ngày cập nhật là 27/05/2006 (vì dãy 16bit tương ứng tại offset 18h là 34BBh = 0011010<u>0101</u>11011b nên ngày = 11011b = 27, tháng = 0101b = 5, năm = 1980 + 0011010b = 2006).*
- *Giờ cập nhật là 9:16:44 (vì dãy bit tương ứng là 4A16h = 01001<u>01000</u>10110b nên giây = 2 * 10110b = 44, phút = 010000b = 16, giờ = 01001b = 9)*

C.2.2 Cấu trúc entry thư mục trên Windows*

Tên tập tin theo khuôn dạng 8.3 rõ ràng là rất khó dùng, khó thể hiện được nội dung tương ứng của tập tin (không chỉ bởi số ký tự quá ít mà còn ở chỗ bảng mã cho ký tự chỉ là bảng mã ASCII 1 byte cũng gây giới hạn). Do đó, đến năm 1994 - khi không gian đĩa đã có sự tăng trưởng lớn, Microsoft đã cải tiến lại thiết kế cho phần tên này : định dạng LFN (Long File Name) ra đời.

LFN sử dụng bảng mã Unicode và có chiều dài lên tới 254 ký tự. Tuy nhiên để tương thích với DOS, các entry LFN vẫn phải có 32 byte giống như entry DOS. Các thông tin cơ bản trên entry của DOS vẫn phải giữ nguyên, những thông tin bổ sung được đưa vào phần 10 byte dành riêng hoặc nơi khác.

Windows giải quyết tên file dài bằng cách rút phần tên dài này thành tên ngắn có đúng 8 ký tự ASCII (và phần tên mở rộng 3 ký tự) rồi lưu vào entry giống như thiết kế trên DOS, phần tên dài được cắt thành từng đoạn (mỗi đoạn 13 ký tự dạng UTF16 – chiếm 26 byte) và lưu mỗi đoạn vào một entry.

Như vậy Windows có 2 loại entry thuộc tập tin: entry chính chứa tên ngắn và các thuộc tính, entry phụ chứa tên dài tương ứng. Mỗi tập tin luôn có 1 entry chính, và từ 0 đến 19 (=254/13) entry phụ.

Các entry phụ của tập tin luôn nằm liên tiếp kế trước entry chính theo thứ tự ngược dần lên. Phân bố các entry của một tập tin như sau:

Entry phụ N	**…..**	**Entry phụ 2**	**Entry phụ 1**	**Entry chính**

Entry trống trên Windows cũng giống như trên DOS, gồm 2 loại:

- Entry 0: byte đầu mang giá trị 0 (cả 32 byte cũng đều bằng 0), là entry trống chưa hề sử dụng.
- Entry E5: byte đầu là E5h, đã từng là entry chính hoặc entry phụ của tập tin nhưng đã bị xóa.

Khi Windows tìm entry trống để cấp sẽ tìm entry 0 trước, nếu không có mới tìm entry E5 (để có thể phục hồi tập tin bị xóa). Khi tìm được một entry 0 thì chắc chắn những entry kế sau cũng là entry 0.

Cấu trúc entry chính :

Offset (hex)	Số byte	Kiểu lưu trữ	Ý nghĩa
[0	8	Chuỗi ký tự ASCII	Tên chính /tên ngắn - lưu bằng mã ASCII
8	3	Chuỗi ký tự ASCII	Tên mở rộng – mã ASCII
B	1	Chuỗi bit	Thuộc tính trạng thái (0.0.A.D.V.S.H.R)
C	1	Chuỗi byte	Dành riêng
D	3	Số nguyên không dấu	Giờ tạo (miligiây:7b; giây:6b; phút:6b; giờ:5b)
10	2	Số nguyên không dấu	Ngày tạo (ngày: 5b; tháng: 4b; năm-1980: 7b)
12	2	Số nguyên không dấu	Ngày truy cập gần nhất (lưu như trên)
14	2	Số nguyên không dấu	Cluster bắt đầu – phần Word (2Byte) cao
16	2	Số nguyên không dấu	Giờ sửa gần nhất (giây/2:5b; phút:6b; giờ:5b)
18	2	Số nguyên không dấu	Ngày cập nhật gần nhất (lưu như trên)
1A	2	Số nguyên không dấu	Cluster bắt đầu – phần Word thấp
1C	4	Số nguyên không dấu	Kích thước của phần nội dung tập tin (theo byte)

Phần tên ngắn của entry chính được rút từ tên dài theo qui tắc: lấy 6 ký tự ASCII khác khoảng trắng đầu tiên trong tên dài đổi thành chữ hoa & gắn thêm ký tự "~" cùng một con số 1 chữ số, sao cho không trùng với những tên ngắn đang hiện diện trên bảng thư mục đó. Trong trường hợp không thể con số 1 chữ số nào để không trùng thì sẽ lấy 5 ký tự gắn với "~" và một con số 2 chữ số, nếu vẫn không được thì cứ giảm số ký tự đi và tăng số chữ số lên.

Ví dụ: tạo trên một thư mục 2 file có tên là "Bai tap thuc hanh.DOC" và "Bai tap ly thuyet.DOC" thì 2 tên ngắn tương ứng có thể là "BAITAP~1.DOC" và "BAITAP~2.DOC".

Trên entry LFN thì Cluster bắt đầu được biểu diễn bằng một số nguyên 4 byte, nhưng trường Cluster bắt đầu của entry DOS chỉ có 2 byte và 2 byte kế trước đã dùng để lưu trữ thông tin khác nên Windows phải lưu 2 byte cao của con số nguyên này vào offset khác .

Ví dụ: cluster bắt đầu là 2006 = 000007D6h thì word cao là 0 và word thấp là 7D6h; cluster bắt đầu là 12345678 = 00BC614Eh thì word cao là 00BCh và word thấp là 614Eh.

Cấu trúc entry phụ :

Offset	Số byte	Kiểu lưu trữ	Ý nghĩa
0	1	Số nguyên không dấu	Thứ tự của entry (bắt đầu từ 1)
1	10d	Chuỗi ký tự UniCode	5 ký tự UniCode – bảng mã UTF16
B (11d)	**1**	**Số nguyên không dấu**	**Dấu hiệu nhận biết - luôn là 0Fh**
C (12d)	1	Số nguyên không dấu	Byte dự trữ (luôn bằng 0)
D (13d)	1	Số nguyên không dấu	CheckSum
E (14d)	12d	Chuỗi ký tự UniCode	6 ký tự UniCode kế tiếp
1A (26d)	2	Chuỗi byte	Dự trữ (luôn bằng 0)
1C (28d)	4	Chuỗi ký tự UniCode	2 ký tự UniCode kế tiếp

Các ký tự được lưu bằng bảng mã Unicode (dựng sẵn, 2 byte) nên có thể biểu diễn được các ký tự của mọi quốc gia chứ không hạn chế như bảng mã ASCII 1 byte.

Số thứ tự của các entry phụ được lưu ở 6 bit thấp của byte đầu tiên. Dấu hiệu để biết entry cuối cùng trong dãy entry phụ là ở 2 bit cao nhất: các entry phụ luôn có 2 bit này là 00 nhưng entry phụ cuối cùng là 01.

Ở entry phụ cuối cùng, nếu ký tự cuối của tên không nằm ở offset cuối thì sau đó là ký tự NULL (mã 0000h), và các ký tự kế tiếp còn lại đều là FFFFh.

Byte CheckSum được tính từ các ký tự lưu trên phần tên ngắn ở entry chính theo thuật toán:

Bước 1: Sum = mã ASCII của ký tự đầu

Bước 2: Quay phải Sum 1 bit

Bước 3: Cộng dồn mã ký tự kế tiếp vào Sum

Bước 4: Nếu chưa xét tới ký tự cuối cùng thì quay lại bước 2.

Trong C/C++, các bước trên có thể mã hóa như sau:

```
for (sum = i = 0; i < 11; i++)  sum = (((sum & 1) << 7) | ((sum & 0xFE) >> 1)) + name[i] ;
```

Ví dụ, dãy byte sau tương ứng với các entry của 2 tập tin trong một bảng thư mục (phần được gạch dưới là các entry chính):

```
42 79 00 65 00 74 00 20 00 2D 00 0F 00 14 20 00  By.e.t. .-.... .
43 00 44 00 2E 00 64 00 6F 00 00 00 63 00 00 00  C.D...d.o...c...
01 44 00 65 00 20 00 74 00 68 00 0F 00 14 69 00  .D.e. .t.h....i.
20 00 4C 00 79 00 20 00 74 00 00 00 68 00 75 00   .L.y. .t...h.u.
44 45 54 48 49 4C 7E 31 44 4F 43 20 00 2D 0C 4A  DETHIL~1DOC .-.J
BB 34 BB 34 01 00 16 4A BB 34 14 00 00 00 01 00  »4»4...J»4......
43 61 00 6D 00 20 00 32 00 30 00 0F 00 31 30 00  Ca.m. .2.0...10.
36 00 00 00 FF FF FF FF FF FF 00 00 FF FF FF FF  6...ÿÿÿÿÿÿ..ÿÿÿÿ
02 79 00 65 00 74 00 20 00 2D 00 0F 00 31 20 00  .y.e.t. .-...1 .
48 00 44 00 48 00 20 00 2D 00 00 00 20 00 6E 00  H.D.H. .-... .n.
01 44 00 65 00 20 00 74 00 68 00 0F 00 31 69 00  .D.e. .t.h...1i.
20 00 6C 00 79 00 20 00 74 00 00 00 68 00 75 00   .l.y. .t...h.u.
44 45 54 48 49 4C 7E 32 20 20 20 10 00 2A 2D 46  DETHIL~2   ..*-F
BB 34 BB 34 00 00 2E 46 BB 34 2D 00 00 00 00 00  »4»4...F»4-.....
```

Từ các bảng cấu trúc entry trên, ta có thể suy ra các thông tin về 2 tập tin này như sau:

	Tập tin thứ nhất	***Tập tin thứ hai***
Tên ngắn	*DETHIL~1.DOC*	*DETHIL~2*
Tên dài	*De thi Ly thuyet – CD.doc*	*De thi ly thuyet - HDH - nam 2006*
Kích thước	*65536 byte*	*0*
Loại tập tin	*tập tin bình thường*	*Tập tin thư mục*
Trạng thái	*Archive*	*Không có*
Cluster bắt đầu	*00010014h = 65556*	*0000002Dh = 45*
Thời điểm tạo	*27/05/2006, 9:16:24*	*27/05/2006, 8:49:26*
Thời điểm cập nhật	*27/05/2006, 9:16:44*	
Thời điểm truy cập	*27/05/2006*	

Khi xóa tập tin thì entry tương ứng phải được chuyển sang trạng thái trống, nhưng không phải 32 byte của entry được đổi qua giá trị 0 mà chỉ có byte đầu tiên được gán thành E5 (tất cả các ký tự của tên tập tin đều không thể có mã là 0 hoặc E5). Để biết một entry có trống hay không ta chỉ cần nhìn giá trị byte đầu tiên: nếu khác 0 và E5 thì đó không phải là entry trống.

Mỗi khi tìm entry trống để sử dụng hệ thống tìm theo thứ tự từ đầu trở đi nên nếu gặp một entry 0 thì tất cả các entry phía sau cũng là những entry 0 (entry trống chưa từng được sử dụng).

D. Bảng SDET (Sub Directory Entry Table)

Mỗi thư mục con được lưu trữ giống như một tập tin bình thường. Nội dung của tập tin thư mục này là một dãy entry, các entry chứa tên & thuộc tính của những tập tin và thư mục con giống y như các entry trên RDET. Trên volume luôn có đúng một RDET nhưng có thể có rất nhiều SDET, và cũng có thể không có bảng SDET nào. Nội dung RDET của volume FAT nằm trên một dãy sector liên tiếp ở vùng system còn nội dung SDET thì nằm trên các cluster của vùng Data và có thể không liên tiếp.

Mỗi SDET luôn có 2 entry '.' và '..' ở đầu bảng mô tả về chính thư mục này và thư mục cha của nó. Vì SDET luôn chiếm trọn cluster nên thuộc tính kích thước tập tin trên entry tương ứng với thư mục sẽ không cần sử dụng và luôn được DOS gán là 0.

Như vậy số entry trong một SDET vừa mới tạo sẽ là $S_C/32$ (S_C là kích thước cluster), các entry từ vị trí thứ ba trở đi đều là entry trống (32 byte đều mang giá trị 0). Nếu chép vào trong thư mục con này nhiều hơn ($S_C/32$ -2) tập tin thì bảng SDET hiện tại không đủ số entry để quản lý, và kích thước SDET sẽ được hệ thống cho tăng thêm bằng nối 1 cluster trống vào, nội dung các entry phát sinh thêm sẽ đưa vào cluster mới này (phần còn lại của cluster lưu các entry trống).

E. Boot Sector

Các thông số quan trọng trong Boot Sector của vol dạng FAT như sau:

Offset (hex)	Số byte	Kiểu lưu trữ	Ý nghĩa
0	3	Chuỗi 24 bit	Lệnh nhảy đến đầu đoạn mã Boot (qua khỏi vùng thông số vol)
3	8	Chuỗi ký tự ASCII	Tên công ty /version của HĐH
B	**2**	**Số nguyên không dấu**	**Số byte của sector** (thường là 512)
D	**1**	**Số nguyên không dấu**	**Số sector của cluster (S_C)**
E	**2**	**Số nguyên không dấu**	**Số sector trước bảng FAT (S_B)**
10	**1**	**Số nguyên không dấu**	**Số lượng bảng FAT (N_F)**, thường là 2
11	**2**	**Số nguyên không dấu**	**Số Entry của RDET (S_R)**, thường là 512 với FAT16
13	**2**	**Số nguyên không dấu**	**Số sector của volume (S_V)**, bằng 0 nếu S_V > 65535
15	1	Số nguyên không dấu	Kí hiệu loại volume
16	**2**	**Số nguyên không dấu**	**Số sector của FAT (S_F)**

18	2	Số nguyên không dấu	Số sector của track
1A	2	Số nguyên không dấu	Số lượng đầu đọc (side)
1C	4	Số nguyên không dấu	Khoảng cách từ nơi mô tả vol đến đầu vol
20	**4**	**Số nguyên không dấu**	**Kích thước volume (nếu số 2 byte tại offset 13h là 0)**
24	1	Số nguyên không dấu	Ký hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
25	1	Chuỗi byte	Dành riêng
26	1	Số nguyên không dấu	Ký hiệu nhận diện HĐH
27	4	Số nguyên không dấu	SerialNumber của Volume
2B	B	Chuỗi ký tự ASCII	Volume Label
36	**8**	**Chuỗi ký tự ASCII**	**Vol ở dạng FAT12 /FAT16 nếu chuỗi là "FAT12__" / "FAT16__"**
3E	1CF	Chuỗi byte	Đoạn chương trình Boot nạp tiếp HĐH khi khởi động máy
1FE	2	Chuỗi byte	Dấu hiệu kết thúc BootSector (luôn là 55h, AAh)

Ví dụ, nếu 128 byte đầu của Boot Sector như sau:

```
EB 3C 90 4D 53 57 49 4E 34 2E 31 00 02 10 01 00
02 00 02 00 00 F8 FF 00 3F 00 FF 00 3F 00 00 00
C2 EE 0F 00 80 00 29 DE 1C 49 15 20 20 20 20 20
20 20 20 20 20 20 46 41 54 31 36 20 20 20 33 C9
8E D1 BC F0 7B 8E D9 B8 00 20 8E C0 FC BD 00 7C
38 4E 24 7D 24 8B C1 99 E8 3C 01 72 1C 83 EB 3A
66 A1 1C 7C 26 66 3B 07 26 8A 57 FC 75 06 80 CA
02 88 56 02 80 C3 10 73 EB 33 C9 8A 46 10 98 F7
```

thì ta có thể suy ra thông tin về các thành phần như sau:

** 2 byte tại offset 0B là: 00, 02*
=> Số byte trên mỗi sector của vol là: 0200h = 512d (byte)
** Giá trị của byte tại offset 0D là: 10*
=> Số sector trên mỗi cluster của vol là: S_C = 10h = 16d (sector)
** 2 byte tại offset 0E là: 01, 00*
=> Số sector trước vùng FAT là: S_B = 0001h = 1d (sector)
** Giá trị của byte tại offset 10 là: 02*
=> Số bảng FAT của vol là: N_F = 02h = 2d (bảng)
** 2 byte tại offset 11 là: 00, 02*
=> Số entry trên bảng RDET là: 0200h = 512d (entry)
*=> Kích thước bảng RDET là: S_R = 512*32/512 = 32 (sector).*
** 2 byte tại offset 16 là: FF, 00*
=> Kích thước bảng FAT là: S_F = 00FFh = 255d (sector)
** 4 byte tại offset 20 là: C2, EE, 0F, 00*
=> Tổng số sector trên vol là: S_V = 000FEEC2h = 1044162d (vì 2 byte tại offset 13h đều là 00 nên S_V được lấy ở 4 byte tại offset 20h)

Từ các thông số trên ta có thể tính ra được kích thước của vùng hệ thống:

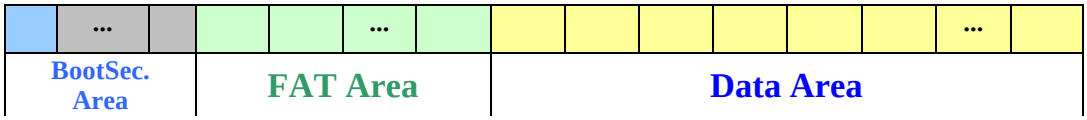
$$S_S = S_B + N_F * S_F + S_R = 1 + 2*255 + 32 = 543 \text{ (sector)}$$

Vậy vùng dữ liệu bắt đầu tại sector 543, và cluster 2 sẽ chiếm 16 sector từ 543 đến 558, cluster 3 sẽ từ 559 đến 574. Tổng quát, cluster K sẽ chiếm 16 sector bắt đầu tại sector có chỉ số 543 + 16(K-2)*

3.2.3. ĐỊNH DẠNG FAT32

Khi bắt đầu phát sinh các volume có kích thước lớn thì kiến trúc FAT không còn dùng được (vì chỉ có thể quản lý tối đa 65518 cluster, mỗi cluster tối đa 128 sector, tức dung lượng volume chỉ được phép tối đa là gần 4GiB). Kiến trúc FAT32 được đưa ra chủ yếu để khắc phục yếu điểm này.

Các thành phần trên FAT32 có thứ tự phân bố như sau:



Vùng Data cũng là dãy cluster đánh chỉ số bắt đầu từ 2 như FAT, các bảng SDET cũng có kiến trúc giống như cũ. Khác biệt lớn có thể nhận thấy trước mắt là ta không còn thấy RDET trên vùng System, vì RDET lúc này được coi như một SDET – và vì vậy được lưu trên vùng Data (cluster bắt đầu của RDET được lưu bằng số nguyên 4 byte ở offset 2Ch trên Boot sector).

Vùng FAT của volume FAT32 cũng thường có 2 bảng, cơ chế quản lý cluster cũng giống như FAT12/FAT16. Nhưng như tên gọi, mỗi phần tử của bảng FAT32 sẽ có 32bit, và nhờ vậy mà phạm vi biểu diễn của mỗi phần tử lớn hơn hẳn so với loại FAT dùng kích thước phần tử 12bit hoặc 16bit, từ đó có thể quản lý được một lượng cluster lớn hơn hẳn.

Giá trị của phần tử trên bảng FAT32 được qui ước như sau:

Giá trị của phần tử	Trạng thái	Diễn giải
0	**FREE**	Cluster tương ứng Trống
0FFFFFF7	**BAD**	Cluster tương ứng Hư
0FFFFFFF	**EOF**	Cluster cuối của file
2 .. 0FFFFFEF	**USED**	Cluster chứa nội dung file, cluster kế sau = <giá trị phần tử>

Boot Sector của hệ thống FAT32

Boot Sector của FAT (FAT12 /FAT16) và FAT32 đều tổ chức thông tin thành 6 vùng theo thứ tự như sau:

i. Jump Code: lệnh nhảy qua khỏi vùng chứa thông số, tới vùng Bootrap Code chứa các lệnh khởi tạo & nạp HĐH. Lệnh này nằm ngay đầu Boot sector (offset 0) & chiếm 3 byte.

ii. OEM ID: Original Equipment Manufacture ID - nơi sản xuất HĐH. Phần này nằm kế tiếp - tại offset 3 & chiếm 8 byte.

iii. BPB: BIOS Parameter Block – khối tham số BIOS, chứa các thông tin quan trọng của vol. Nằm kế sau vùng OEM_ID - tại offset Bh, và chiếm 19h (25) byte nếu là hệ thống FAT, chiếm 35h (53) byte nếu là hệ thống FAT32.

iv. BPB mở rộng: chứa những tham số bổ sung. Vùng này chiếm 1Ah (26) byte và nằm tại offset 24h (36) nếu là hệ thống FAT, tại offset 40h (64) nếu là hệ thống FAT32.

v. Bootrap Code: đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy. Với FAT phần này bắt đầu tại offset 3Eh (62) & chiếm 1CFh (448) byte, với FAT32 bắt đầu tại offset 5Ah (90) & chiếm 1A4h (420) byte.

vi. Dấu hiệu kết thúc: xác định tính hợp lệ của BootSector, là 2 byte cuối cùng của sector (offset 1FEh & 1FFh) và luôn có giá trị là 55h, AAh. (2 byte cuối của Master Boot Sector cũng có dấu hiệu nhận diện là 55h & AAh như vậy).

Trong vùng Boot Sector của FAT32 còn có một sector chứa các thông tin hỗ trợ cho việc xác định tổng số cluster trống & tìm kiếm cluster trống được hiệu quả, và một sector chứa bản sao của Boot sector. Vị trí của các sector này cũng được biểu diễn trong BPB.

Sau đây là tổ chức thông tin chi tiết của Boot Sector theo FAT32:

Offset	Số byte	Kiểu lưu trữ	Nội dung
0	3	Số nguyên	Jump_Code: lệnh nhảy qua vùng thông số (như FAT)
3	8	Chuỗi ký tự	OEM_ID: nơi sản xuất – version, thường là "MSWIN4.1"
B	**2**	**Số nguyên**	**Số byte trên Sector, thường là 512 (như FAT)**
D	**1**	**Số nguyên**	**S_C: số sector trên cluster (như FAT)**
E	**2**	**Số nguyên**	**S_B: số sector thuộc vùng Bootsector (như FAT)**
10	**1**	**Số nguyên**	**N_F: số bảng FAT, thường là 2 (như FAT)**
11	2	Chuỗi byte	Không dùng, thường là 0 (số entry của RDET – với FAT)

13	2	Chuỗi byte	Không dùng, thường là 0 (số sector của vol – với FAT)
15	1	Số nguyên	Loại thiết bị (F8h nếu là đĩa cứng - như FAT)
16	2	Chuỗi byte	Không dùng, thường là 0 (số sector của bảng FAT – với FAT)
18	2	Số nguyên	Số sector của track (như FAT)
1A	2	Số nguyên	Số lượng đầu đọc (như FAT)
1C	4	Số nguyên	Khoảng cách từ nơi mô tả vol đến đầu vol (như FAT)
20	**4**	**Số nguyên**	**S_V: Kích thước volume** (như FAT)
24	**4**	**Số nguyên**	**S_F: Kích thước mỗi bảng FAT**
28	2	Số nguyên	bit 8 bật: chỉ ghi vào bảng FAT active (có chỉ số là 4 bit đầu)
2A	2	Số nguyên	Version của FAT32 trên vol này
2C	**4**	**4**	**Cluster bắt đầu của RDET**
30	**2**	**2**	**Sector chứa thông tin phụ (về cluster trống), thường là 1**
32	**2**	**2**	**Sector chứa bản lưu của Boot Sector**
34	C	Chuỗi byte	Dành riêng (cho các phiên bản sau)
40	1	Số nguyên	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Số nguyên	Dành riêng
42	1	Số nguyên	Kí hiệu nhận diện HĐH
43	4	Số nguyên	SerialNumber của Volume
47	B	Chuỗi ký tự	Volume Label
52	**8**	**Chuỗi ký tự**	**Loại FAT, là chuỗi "FAT32 "**
5A	1A4	Chuỗi byte	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	2	Chuỗi byte	Dấu hiệu kết thúc BootSector /Master Boot (luôn là 55h, AAh)

Tổ chức trong sector chứa thông tin phụ được mô tả ở offset 30h:

Offset	Số byte	Nội dung
00h	4	Dấu hiệu nhận biết (phải là 52h, 52h, 61h, 41h)
04h	480	Chưa sử dụng
1E4h	4	Dấu hiệu nhận biết có thông tin (phải là 61417272h)
1E8h	4	Số lượng cluster trống (là -1 nếu không biết)
1ECh	4	Vị trí bắt đầu để tìm cluster trống (là -1 nếu không biết)
1F0h	12	Chưa sử dụng
1FCh	4	Dấu hiệu kết thúc (phải là 00h, 00h, 55h, AAh)

Ví dụ, với vol có 128 byte đầu của Boot Sector như sau:

```
EB 58 90 4D 53 57 49 4E  34 2E 31 00 02 08 26 00 ëX□MSWIN4.1...&.
02 00 00 00 00 F8 00 00  3F 00 F0 00 3F 00 00 00 .....ø..?.ð.?...
21 6C 9C 00 07 27 00 00  00 00 00 00 02 00 00 00 !lœ..'..........
01 00 06 00 00 00 00 00  00 00 00 00 00 00 00 00 ................
80 00 29 07 1D 04 1A 50  31 46 33 32 2D 35 30 20 €.)....P1F32-50
20 20 46 41 54 33 32 20  20 20 33 C9 8E D1 BC F4   FAT32   3ÉŽÑ¼ô
7B 8E C1 8E D9 BD 00 7C  88 4E 02 8A 56 40 B4 08 {ŽÁŽÙ½.|ˆN.ŠV@´.
CD 13 73 05 B9 FF FF 8A  F1 66 0F B6 C6 40 66 0F Í.s.¹ÿÿŠñf.¶Æ@f.
```

thì căn cứ vào bảng tham số vol của FAT32 ta có thể suy ra các thông tin:

S_C = <Số nguyên 1 byte tại offset Dh> = 08h = 8d (sector).

S_B = <Số nguyên 2 byte tại offset Eh> = 0026h = 48d (sector).

N_F = <Số nguyên 1 byte tại offset 10h> = 02h = 2.

S_V = <Số nguyên 4 byte tại offset 20h> = 009C6C21h = 102512976d (sector).

S_F = <Số nguyên 4 byte tại offset 24h> = 00002707h = 9991d (sector).

Cluster bắt đầu RDET = <Số nguyên 4 byte tại offset 2Ch> = 00000002h = 2

Sector chứa thông tin phụ = <Số nguyên 2 byte tại offset 30h> = 0001h = 1

Sector chứa bản sao BootSector = <Số 2 byte tại offset 32h> =0006h = 6

……

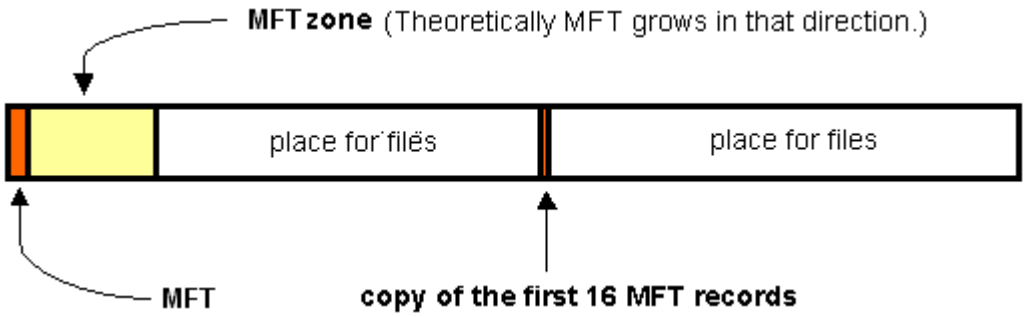
3.3. CÁC KIẾN TRÚC HỆ THỐNG TẬP TIN CAO CẤP

3.3.1. NTFS

A. Sơ lược về partition format bằng định dạng NTFS

Cũng như mọi hệ thống khác, NTFS chia tất cả vùng dữ liệu của nó thành các khối cluster. NTFS hỗ trợ hầu như tất cả các kích thước của cluster, từ 512 byte đến 64 Kb.

Thông thường, vol NTFS được chia thành 2 phần, 12% đầu tiên của đĩa được gọi là MFT, nơi chứa MFT metafile. Phần còn lại là phần Data để chứa dữ liệu.



Khi vol chưa chứa dữ liệu, tất cả các sector trên nó đều trống kể cả vùng MFT. Tổ chức của MFT như sau: khi một file không thể ghi vào vùng dữ liệu được thì MFT tự động co lại để có chỗ ghi file, khi vùng dữ liệu được giải phóng nó lại phình ra trở lại. Tuy nhiên có một trở ngại là có thể có những file vẫn còn sót lại trong MFT gây ra hiện tượng phân mảnh trong MFT.

B. MFT và cấu trúc của nó:

Hệ thống NTFS xem mỗi thành phần của nó là một file, và file quan trọng nhất trong hệ thống là MFT (Master File Table). Nó chịu trách nhiệm quản lý tất cả các file và cả chính nó trong đĩa. MFT được phân chia ra thành các record (thường có kích thước là 1KB) và 1 record tương ứng với một vài file. 16 file đầu không cho hệ điều hành có thể truy cập và có tên là metafile. Metafile là phần duy nhất của đĩa có vị trí cố định. Có một bản copy 16 record đầu tiên để tăng độ tin cậy của hệ thống (được lưu ở ngay chính giữa đĩa).

Có ít nhất một entry của MFT cho mỗi file trong volume NTFS, kể cả chính nó. Tất cả các thông tin về file như kích thước, ngày tạo, quyền truy cập và nội dung dữ liệu được lưu trữ trong các MFT entry, hoặc ở vùng dữ liệu ngoài MFT mà MFT entry trỏ tới. Nghĩa là khi có một số thuộc tính nào đó của file không ghi đủ vào MFT entry thì nó sẽ được lưu trong một hay vài cluster của vùng dữ liệu.

C. Metafile:

16 file đầu tiên của MFT. Mỗi file có một trách nhiệm về một phương diện nào đó của hệ điều hành. Vì chỉ có 16 MFT đầu tiên nên kích thước rất nhỏ gọn, có khả năng chịu lỗi cao. Ví dụ như nếu có lỗi xảy ra trên bảng FAT của hệ thống quản lý bằng FAT thì sẽ gây ra lỗi đĩa. Tuy nhiên, nếu có lỗi vậy lý của đĩa xảy ra trên vùng MFT thì nó sẽ tự động tránh chỗ đó ra (dĩ nhiên ngoại trừ 16 MFT đầu tiên).

Metafile nằm trong vùng NTFS disk root directory, tên của chúng bắt đầu bằng kí tự "$". Dưới đây là một số metafile được sử dụng thường xuyên, quan trọng và chức năng được chỉ định của chúng:

$MFT	Chính MFT file
$MFTmirr	Vị trí của bảng copy 16 MFT đầu tiên ở giữa partition.
$LogFile	Hỗ trợ ghi file nhật kí.
$Volume	Thông tin quản lý - nhãn đĩa, phiên bản của file hệ thống v.v..

$AttrDef	Danh sách các file thuộc tính trên đĩa.
$.	Root directory
$Bitmap	Vùng còn trống trên đĩa.
$Boot	boot sector (ở những partition có thể boot được)
$Quota	Dung lượng đĩa mà người dùng có thể sử dụng
$Upcase	File – bảng lưu sự phù hợp tương ứng giữa chữ hoa và chữ thường. điều này là cần thiết vì NTFS sử dụng bảng mã ASCII chứa rất nhiều kí tự. việc không phân biệt chữ thường chữ hoa sẽ tiết kiệm được thời gian tìm kiếm.

D. File

Tất cả các thông tin về file đều được lưu trữ trong MFT (ngoại trừ chính nó): tên file, kích thước của nó, các thành phần phân mảnh trên đĩa của file đó. Nếu MFT không đủ để chứa thông tin, sẽ có nhiều MFT record được sử dụng và không bắt buộc chúng phải liền nhau.

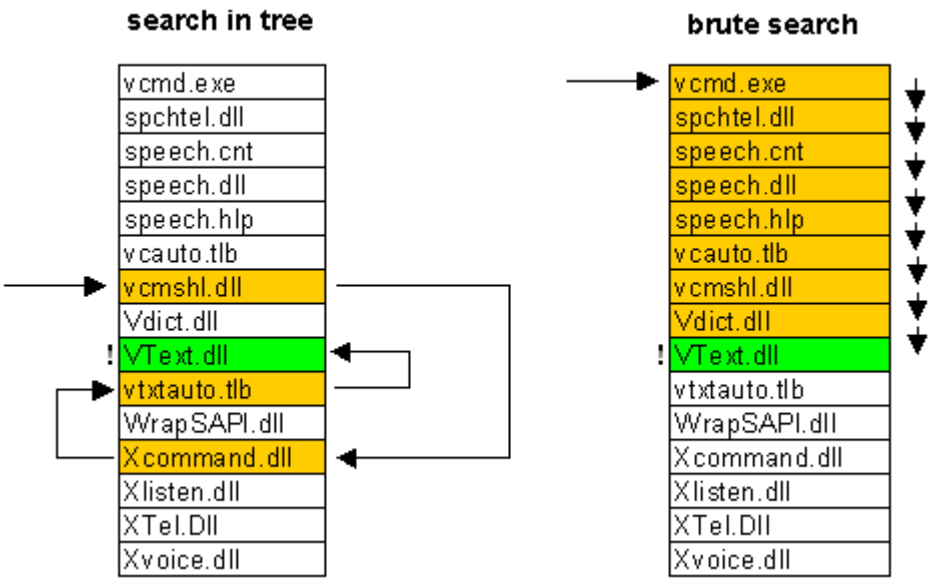
Nếu một file không có dữ liệu bên trong nó thì đĩa sẽ không lưu nó mà chỉ lưu trong MFT. Nếu một file có kích thước rất nhỏ, để tránh lãng phí đĩa, nó sẽ được lưu trực tiếp lên MFT.

Mỗi file trong NTFS chỉ là một lớp trừu tượng, nó không có dữ liệu, nó đơn giản chỉ là một lớp. Mỗi một lớp chứa tất cả các thuộc tính của file đó. Tên file được lưu bằng Unicode và dài tối đa 255 kí tự.

E. Thư mục:

Thư mục trong NTFS là một file đặc biệt. Nó được chia ra làm 2 phần. Một trong số chúng chứa tên file, những thuộc tính căn bản và chỉ mục đến phần tử MFT chứa thông tin đầy đủ về nó. Cấu trúc bên trong của thư mục là cấu trúc cây nhị phân. Điều này làm cho việc tìm kiếm tên một thư mục nào đó được nhanh chóng hơn so với kiểu tìm kiếm từ trên xuống dưới của FAT.

Ví dụ:



(Linux-NTFS, http://www.linux-ntfs.org/content/view/104/43/)

3.3.2. exFAT

(NTFS, FAT,exFAT http://ntfs.com/)

Các Thực nghiệm

TN#2.1	Phân biệt chính xác các thiết bị lưu trữ phổ dụng, khảo sát các hình thức kết nối để truy xuất DL
TN#2.2	Vệ sinh các điểm tiếp xúc, trợ nguồn, kết nối trực tiếp
TN#2.3	Mở và khảo sát đầu đọc, mâm đĩa và các phụ kiện bên trong ổ cứng – cách thay thế

Các Bài tập :

BTVN#2.1	Lập bảng so sánh các đặc điểm chính của các thiết bị trên

BTVN#2.2 (0.2điểm)	**Lập bảng so sánh chi tiết HDD, SSD và SSHD**
BTVN#2.3	Thí nghiệm mở /gắn board HDD (đưa SV đĩa cứng để mang về làm)
BTVN#2.4	Nghiên cứu cách chuyển đổi qua lại giữa sector logic và sector vật lý

Các Đồ án:

ĐAMH#1 (0.8đ)	**Tìm hiểu và trình bày về tổ chức Partition trên đĩa. So sánh các ưu khuyết điểm giữa MBR và GPT**
ĐAMH#2 (0.8đ)	**Viết chương trình truy xuất trực tiếp sector logic /vật lý, trình bày các thông tin lấy được từ bảng Parttion /BootSector (nếu có thể thì thực hiện giao diện tương tự WinHex) và tạo Image cho đĩa cần cứu chữa (chương trình cần có khả năng tự chạy tiếp phần việc còn dở dang khi có sự cố gây gián đoạn – ví dụ như mất điện)**