HỆ THỐNG QUẢN LÝ TẬP TIN

1. ĐẶT VẤN ĐỀ

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 (HĐH), ...)*

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 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 HĐH, cụ thể là bởi thành phần hệ thống quản lý tập tin.

2. XÂY DỰNG CÁC KHÁI NIỆM CẦN THIẾT

2.1 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.

2.2 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).

2.3 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 đĩa mềm, đĩa cứng bỏ túi, đĩa CD, ...) 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.

3. THIẾT KẾ MÔ HÌNH TẬP TIN & THƯ MỤC

3.1 Mô hình thuộc tính

3.1.1 Tập tin

* Nội dung tập tin: được lưu trữ dưới dạng nén hay không nén; là dãy byte tuần tự, dãy các record chiều dài cố định hay theo cấu trúc cây.

* Tên tập tin: ngoài phần tên chính ra có phần tên mở rộng hay không, nếu có thì bao nhiêu phần mở rộng, các thông tin chi tiết của từng phần: chiều dài tối đa & tối thiểu / có phân biệt thường hoa / có cho dùng khoảng trắng / có dấu hay không (font tự do hay font cố định?) / các ký tự không được dùng / các ký tự đại diện / các chuỗi không được trùng / có *cho phép trùng tên / giá trị các byte rác (khi tên chưa đạt tới chiều dài tối đa) / byte kết thúc / chiều dài tên / .....*

* Thời điểm tạo: ngày-tháng-năm & giờ-phút-giây lưu thành các trường riêng biệt hay ghép lại thành một dãy bit.

Ví dụ:

- dạng 1: (ngày 18/10/2006 – 15:26:05)

	Ngày	*Tháng*	*Năm*		*Giờ*	*Phút*	*Giây*
	12	**0A**	**D6**	**07**	**0F**	**1A**	**05**
offset	0	1	2	3	4	5	6

- dạng 1: (ngày 18/10/2006 – 15:26:05)

	Ngày Tháng Năm		*Giờ Phút Giây*	
	35	**52**	**13**	**4F**
offset	0	1	2	3

trong đó ngày chiếm 5bit, tháng chiếm 4bit, năm chiếm 7bit (lưu 2 chữ số cuối hoặc giá trị của hiệu <năm>-1980); giờ chiếm 5bit, phút chiếm 6bit, giây chiếm 5bit (lưu giá trị <giây>/2)

- ....

* Thời điểm cập nhật: tương tự.

* Thời điểm sử dụng gần nhất: có thể chỉ lưu ngày, không cần lưu giờ.

* Kích thước phần nội dung tập tin: là số nguyên bao nhiêu byte (còn tùy thuộc nội dung tập tin lưu dưới dạng dãy byte tuần tự hay record).

* Mật khẩu: chuỗi tối đa bao nhiêu ký tự, tối thiểu bao nhiêu ký tự (nếu khác rỗng thì sau khi kiểm tra đúng mới cho truy xuất nội dung tập tin)

* Các thuộc tính trang 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)

… …

3.1.2 Thư mục

Thiết kế mô hình tương tự như trên. Vì thư mục cũng có khá nhiều thuộc tính giống với các thuộc tính của tập tin nên nhiều hệ thống 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.

3.2 Các chức năng

* 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: …

* …

4. TỔ CHỨC HỆ THỐNG TẬP TIN TRÊN KHÔNG GIAN LƯU TRỮ

4.1 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)*

4.2 Cluster

4.2.1 Khái niệm

Đơn vị đọc ghi trên đĩa là sector, nhưng đơn vị lưu trữ nội dung tập tin không phải là sector mà nên là một *cluster* gồm N sector liên tiếp (N≥1).

☞ Mỗi vị trí để lưu giữ nội dung tập tin trong các phân tích trên sẽ là 1 cluster.

☞ Cluster chỉ tồn tại trên vùng dữ liệu (vùng DATA) – nơi chứa nội dung tập tin.

4.2.2 Lý do phát sinh khái niệm Cluster

- Nếu sector trên vùng dữ liệu quá nhiều thì có thể sẽ khó hoặc không quản lý được, khi đó quản lý trên cluster sẽ dễ dàng hiệu quả hơn.

- Nội dung tập tin thường chiếm khá nhiều sector và có thể không liên tục, lưu giữ nội dung theo từng dãy 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.

4.2.3 Hình thức tổ chức

Vol sẽ được chia thành 2 vùng: vùng dữ liệu (DATA) chứa nội dung tập tin và vùng hệ thống (SYSTEM) chứa các thông tin quản lý.

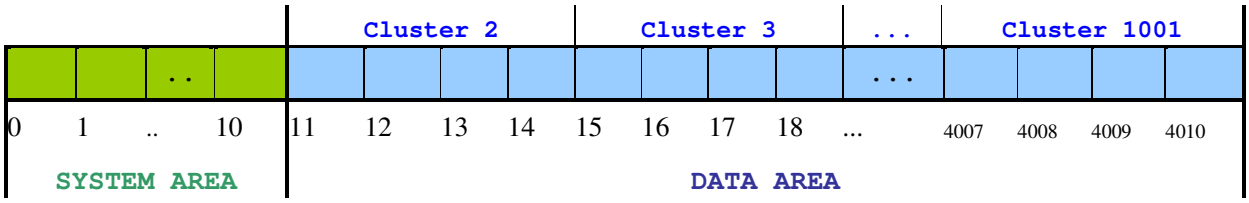
Vùng SYSTEM có kích thước nhỏ hơn nhiều so với vùng DATA và phải truy xuất mỗi khi sử dụng Vol nên thường nằm ngay đầu Vol, phần còn lại thuộc về vùng DATA.

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 được đánh chỉ số là $\mathbf{F_C}$, thì ta sẽ có:

- Số cluster của vol: $\mathbf{[S_D/Sc]}$
- 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)

4.2.4 Kích thước Cluster

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 của Cluster phụ thuộc vào khá 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 sử dụng,… Trên các đĩa cứng hiện tại thì cluster thường có kích thước 4, 8 hoặc 16 sector.

Ví dụ:

- Với đĩa mềm 1.44 MB bình thường, nếu ta cho 1 cluster chiếm 1000 sector thì sẽ rất không hợp lý. Vì khi đó ta chỉ có thể chép được tối đa 2 tập tin vào đĩa, dẫu mỗi tập tin chỉ có kích thước là 1byte! Tuy nhiên nếu ta chỉ toàn chép vào đĩa các tập tin có kích thước gần 1000 sector thì sẽ không còn sự lãng phí nữa.
- Với hệ thống chỉ có thể quản lý tối đa M Cluster, nếu vùng DATA của Vol có 100M sector thì ta phải cho mỗi cluster tối thiểu 100 sector mới có thể quản lý được toàn bộ vùng DATA

4.3 Bảng quản lý Cluster

4.3.1 Khái niệm

- Được đưa ra để xác định danh sách các cluster chứa nội dung của một tập tin và các cluster trống.
- Có thể dùng một bảng chung, cũng có thể tổ chức nhiều bảng – mỗi bảng phục vụ cho một nhu cầu.

- Thường được tổ chức dưới dạng một dãy phần tử - mỗi phần tử là một con số nguyên dùng để quản lý một cluster trên vùng dữ liệu, cho biết cluster tương ứng đang ở trạng thái trống, hư, hay đang chứa nội dung của một tập tin, và cho phép xác định danh sách các cluster chứa nội dung của tập tin.

4.3.2 Lý do phát sinh

Để đưa nội dung tập tin vào vol thì phải xác định các cluster còn trống, để đọc nội dung tập tin trên vol thì phải xác định được danh sách các cluster chứa nội dung tập tin đó. Tuy có thể lưu thông tin quản lý ngay trên cluster nhưng khi đó truy xuất sẽ rất chậm nên nhất thiết phải lập ra bảng này để quản lý truy xuất được nhanh chóng.

4.3.3 Hình thức tổ chức

4.3.3.1 Quản lý cluster hư:

Vấn đề này không phải quan trọng lắm, vì đa số volume không có cluster hư (vật lý). Tuy nhiên với các volume có khả năng phát sinh cluster hư thì cũng nên quản lý, có những cách thiết kế sau:

- Lưu 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.

4.3.3.2 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) lại cho hệ thống hiệu quả hơn.

Tuy có thể tổ chức thông tin trạng thái ngay trên cluster (dùng 1 byte trên cluster để biểu diễn trạng thái), nhưng khi này việc xác định danh sách các cluster trống sẽ rất chậm, vì đầu đọc phải thực hiện quá nhiều thao tác di chuyển /đọc. Ta cần dùng một bảng riêng để quản lý, có những thiết kế như sau:

- *Dùng hình thức bitmap*: sử dụng một dãy byte, trong đó mỗi bit trên dãy quản lý 1 cluster tương ứng. 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 nằm tại byte (K div 8) và là bit (K mod 8) trên byte đó.

- *Quản lý theo dạng chỉ mục*: Để xác định các cluster trống, hư, hay thuộc trạng thái luận lý đặc biệt nào đó (những trạng thái chỉ mang 1 trong 2 giá trị: Có – Không) 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.

4.3.3.3 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 tiếp (danh sách đặc).

- Sử dụng cấu trúc danh sách liên kết

- Sử dụng cấu trúc danh sách liên kết kết hợp chỉ mục (index).

- Sử dụng cấu trúc cây (kết hợp chỉ mục).

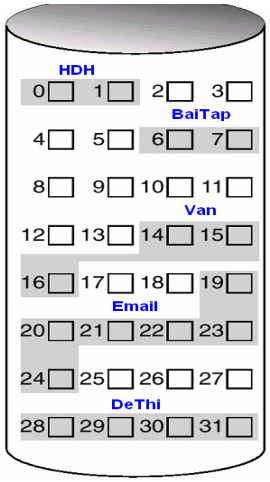
4.3.3.3.1 Lưu trữ nội dung tập tin trên dãy cluster liên tiếp

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.

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

- *HDH: (0, 2006)*
- *BaiTap: (6, 2007)*
- *Van: (14, 2050)*
- *Email: (19, 6000)*
- *DeThi: (28, 4096)*

thì phân bố các cluster của những tập tin đó sẽ như hình vẽ



☞ 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.

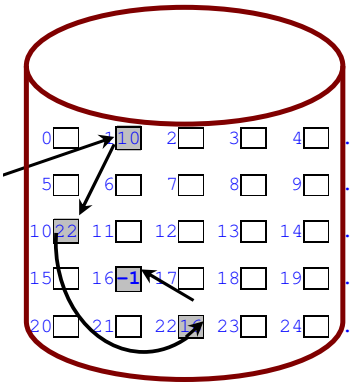
4.3.3.3.2 Sử dụng cấu trúc danh sách liên kết

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. Vì trên thực tế hệ thống phải liên tục thực hiện việc truy xuất nội dung tập tin, nên 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 tập tin)

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 vẽ*



4.3.3.3.3 Sử dụng 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 này ta còn biết được cluster kế tiếp chứa nội dung của tập tin: nếu phần tử K của bảng có giá trị L và L = EOF thì cluster K đã là cluster cuối cùng của tập tin, nếu L ≠ EOF thì phần kế tiếp của nội dung tập tin nằm tại cluster L (quản lý theo dạng danh sách liên kết).

☞ 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 một tập tin (khi biết cluster bắt đầu).

Ví dụ:

+ Nếu 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	13	14	15

EOF						16						
16	17	18	19	20	21	22	23	24	25	26	27	...

+ 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.

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 trạng thái của cluster mang chỉ số K).

Giá trị của các phần tử trên bảng:

Ta cần phải đặt ra 3 giá trị hằng đặc biệt là FREE, BAD và EOF tương ứng với 3 trạng thái cluster: trống, hư, hoặc là cluster cuối của nội dung tập tin, ta cũng có thể đặt thêm một số giá trị đặc biệt nữa cho những trạng thái khác (nếu có) và /hoặc một số giá trị dành riêng cho những nhu cầu trong tương lai. Những giá trị còn lại sẽ

tương ứng với trạng thái cluster đang chứa nội dung tập tin và lúc đó giá trị của phần tử cũng chính là chỉ số của cluster kế tiếp.

Các hằng đặc biệt phải khác với các giá trị có thể có trên chỉ số cluster thì cluster tương ứng mới có thể sử dụng được. Ví dụ, với hình trên, nếu ta đặt hằng EOF = 7 thì tập tin III sẽ bị hiểu lầm là chỉ chiếm đúng mỗi cluster 9 chứ không phải chiếm 4 cluster!

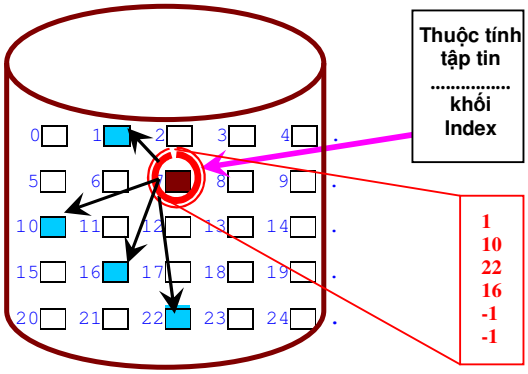
Mỗi phần tử trên bảng nên là một con số nguyên dương để có thể quản lý được nhiều cluster hơn, khi này các hằng đặc biệt nêu trên sẽ không thể là số âm mà phải chiếm những giá trị lớn nhất có thể biểu diễn được, và chỉ số của cluster cuối cùng trên vùng DATA buộc phải nhỏ hơn những giá trị hằng đặc biệt này. Trong trường hợp cluster đầu tiên được đánh chỉ số là $\mathbf{F_C} > 0$ thì cũng có thể sử dụng các giá trị $\mathbf{0..F_C-1}$ cho những hằng đặc biệt trên.

4.3.3.3.4 Sử dụng cấu trúc cây

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.

Ví dụ, nếu tập tin chiếm các cluster là 1, 10, 22, 16 thì sơ đồ lưu trữ có thể như sau:



Kiến trúc I-node của hệ điều hành UNIX:

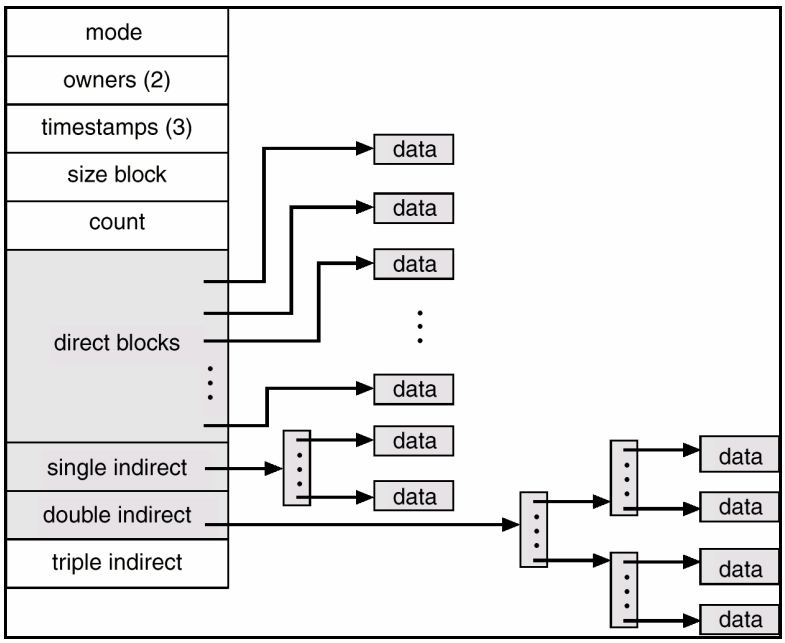
- Mỗi khối có 4KB.
- Mỗi phần tử là một số nguyên 4 byte.
- Trong bảng Triple Indirect có 1024 con trỏ, mỗi con trỏ trỏ đến 1 bảng Double Indirect.
- Trong bảng Double Indirect có 1024 con trỏ, mỗi con trỏ trỏ đến 1 bảng Single Indirect.
- Trong mỗi bảng Single Indirect có 1024 chỉ số cluster trực tiếp.
- Ở I-node gốc có 10 chỉ số cluster trực tiếp.

Từ đó có thể suy ra:

+ Hệ thống dùng cây 1024 nhánh.

+ Số khối tối đa trên hệ thống: 2^{32} (khoảng 4 tỉ)

+ Số khối tối đa của 1 tập tin: $10 + 1024 + 1024^2 + 1024^3$ (khoảng 1 tỉ).



(Tổ chức I-node của UNIX)

4.3.4 Vị trí & Kích thước của Bảng Quản lý Cluster

- Vị trí của bảng: Vì có tầm quan trọng rất lớn & được truy xuất trước tiên mỗi khi truy xuất tập tin, nên cần đặt vào một vị trí cố định và an toàn – đồng thời có thể truy xuất nhanh. Đa số HĐH thiết kế bảng này ở vùng SYSTEM & nằm ngay đầu volume, chỉ sau vùng Boot Sector.

- Kích thước mỗi phần tử: để đơn giản có thể dùng 16bit hoặc 32bit tùy theo vol có kích thước nhỏ hay lớn. Số bit tối thiểu của mỗi phần tử có thể căn cứ vào chỉ số cluster cuối cùng trên vùng DATA và giá trị của các hằng đặc biệt.

- Số phần tử: phụ thuộc vào số cluster trên vùng DATA, không thể tự đặt ra như với những thành phần khác. Và kích thước vùng DATA thì lại phụ thuộc vào kích thước các thành phần còn lại, trong đó có bảng quản lý cluster! Khi đã xác định kích thước các thành phần ta có thể lập được một phương trình bậc nhất 2 ẩn:

<Kích thước DATA> + <Kích thước bảng quản lý cluster> = Constant

Ta có thể giải được bằng cách giả định kích thước của bảng là 1 sector & kiểm chứng lại xem có hợp lý hay không, nếu không thì tăng dần kích thước của bảng cho đến khi hợp lý.

4.3.5 Tổ chức quản lý cluster trên DOS & Windows

4.3.5.1 Hình thức quản lý:

Bảng quản lý cluster trên DOS & Windows 9x được gọi là **FAT** (File Allocation Table), được tổ chức theo hình thức danh sách liên kết kết hợp chỉ mục (trên Windows họ NT có thêm hệ thống **NTFS**).

4.3.5.2 Vị trí & kích thước:

Bảng quản lý cluster rất quan trọng nên DOS & Windows thường có N_F bảng để phòng tránh hư hỏng, số lượng bảng FAT của vol được lưu trữ bằng số nguyên 1 byte tại offset 10h của Boot Sector (đại đa số vol có N_F = 2). Các bảng FAT luôn nằm kế tiếp nhau trên một vùng gọi là vùng FAT.

Vùng FAT nằm kế sau vùng BootSector, nên vị trí bắt đầu của vùng FAT cũng chính là kích thước vùng BootSector (viết tắt là S_B), được lưu trữ bằng 2 byte tại offset E của Boot Sector.

Kích thước mỗi bảng FAT (viết tắt là S_F) được lưu trữ bằng số nguyên 2 byte tại offset 16h của Boot Sector.

4.3.5.3 Các loại FAT:

Trên HĐH DOS mỗi phần tử của bảng FAT được biểu diễn bằng một con số nguyên 12bit hoặc 16bit, bảng FAT tương ứng có tên là **FAT12** hoặc **FAT16**.

Trên HĐH Windows cũng có 2 loại FAT trên và có thêm loại **FAT32** - mỗi phần tử được lưu bằng 32bit. Với Windows thì FAT có nghĩa là FAT12 hoặc FAT16, còn FAT32 được xem là FAT mở rộng.

4.3.5.4 Giá trị của các phần tử trên FAT:

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	FAT32	
Trống	**0**	**0**	**0**	**= FREE**
Hư	**FF7**	**FFF7**	**0FFFFFF7**	**= BAD**
Cluster cuối của file	**FFF**	**FFFF**	**0FFFFFFF**	**= EOF**
Chứa nội dung file, và có cluster kế sau là L	**2 .. FEF**	**2 .. FFEF**	**2..0FFFFFEF**	

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 tồn tại cluster mang chỉ số 0 trên vùng dữ liệu, **DOS & Windows đánh chỉ số của cluster đầu tiên trên vùng dữ liệu là 2** (FC = 2)

Cũng từ những giá trị ở bảng trên, ta có thể suy ra số cluster tối đa mà bảng FAT12 có thể quản lý được là FEE (tức 4078d, chứ không phải là 2^{12} = 4096d), và FAT16 là FFEE (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*

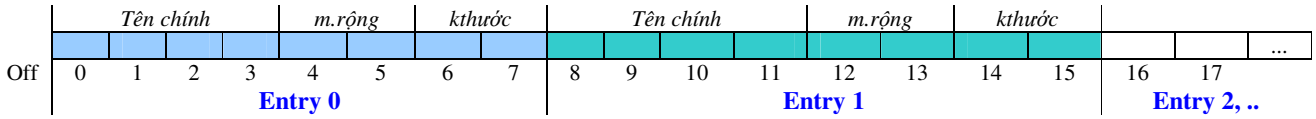
Vì kích thước bảng FAT12 là khá nhỏ (tối đa 12 sector) nên để việc truy xuất một phần tử được đơn giản ta nên đọc toàn bộ các sector trên bảng FAT vào một vùng đệm rồi từ vùng đệm này xây dựng ra một mảng số nguyên 16bit mà mỗi phần tử của mảng mang giá trị của phần tử tương ứng trên bảng FAT. Khi cần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

4.4 Bảng Thư Mục

4.4.1 Khái niệm:

- Là một dãy phần tử (entry), mỗi phần tử chứa tên & các thuộc tính của một tập tin trên vol (hoặc là phần tử trống – chưa thuộc về tập tin nào hết).
- 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).

4.4.2 Lý do phát sinh

Từ phân tích (iii).

4.4.3 Vị trí và Kích thước của Bảng Thư mục

Bảng SDET chắc chắn sẽ nằm ở vùng DATA và không có kích thước cố định, vì thực chất SDET là một tập tin. Với RDET thì một số HĐH qui định phải nằm trên vùng SYSTEM và phải có kích thước cố định (như kiến trúc FAT12/FAT16). Trong trường hợp số entry trên RDET là cố định thì nó sẽ luôn là một hằng số, hoặc được định ra bởi chương trình Format - có thể tùy thuộc vào dung lượng vol nhưng sẽ không quá lớn.

Nếu hệ thống không qui định RDET cố định thì có thể lưu RDET trên vùng DATA giống như SDET, và quản lý nội dung giống như một tập tin bình thường.

Bảng SDET ban đầu sẽ chiếm một 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, và luôn là bội của kích thước cluster.

4.4.4 Tổ chức bảng thư mục của DOS

4.4.4.1 Cấu trúc Entry:

Mỗi entry có 32 byte, chứa các thông tin được tổ chức như sau:

Offset	Số byte	Ý nghĩa
0h (0)	8	Tên chính của tập tin
8h (8)	3	Tên mở rộng
Bh (11)	1	Thuộc tính trạng thái (0.0.A.D.V.S.H.R)
Ch (12)	10	Không dùng
16h (22)	2	Giờ cập nhật tập tin
18h (24)	2	Ngày cập nhật tập tin

1Ah (26)	2	Cluster bắt đầu
1Ch (28)	4	Kích thước tập tin

Hay:

	Tên			*Tt*	*Ko dùng*			*Giờ*		*Ngày*		*C. đầu*		*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

Khi mới định dạng vol, các entry trên RDET đều ở trạng thái trống, và 32 byte trên entry đều có giá trị 0. Mỗi khi chép một tập tin vào thư mục, một entry trống sẽ được sử dụng để chứa các thông tin quản lý tập tin. Các thuộc tính trong entry sẽ có giá trị như sau:

- 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:

		Archive	Directory	VolLabel	System	Hidden	ReadOnly
0	0	x	x	x	x	x	x
7	6	5	4	3	2	1	0

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

+ 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.

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

+ 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.

+ 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.

+ Archive: thuộc tính lưu trữ, cho biết tập tin đã được backup hay chưa (bằng lệnh backup của HĐH), đây là thuộc tính đã 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. 3 thông tin này được lưu trong 2 byte theo hình thức: giờ lưu bằng 5 bit đầu tiên, phút chiếm 6 bit kế tiếp và 5 bit còn lại chứa giá trị giây/2.

- 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).

Khi ta xóa tập tin, entry tương ứng phải được chuyển sang trạng thái trống, nhưng không phải là 32 byte của entry được chuyển sang giá trị 0 mà chỉ có byte đầu tiên được đổi thành E5. Vì tất cả các ký tự của tên tập tin đều không thể có mã là 0 hoặc E5 nên để biết một entry có trống hay không ta chỉ cần nhìn giá trị byte đầu tiên của entry: 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 ta gặp một entry trống có byte đầu là 0 thì tất cả các entry phía sau cũng là những entry trống ở dạng chưa từng được sử dụng (32 byte trên entry đều là 0).

4.4.4.2 RDET:

Luôn nằm ở cuối vùng SYSTEM, kế sau vùng FAT. Tức vị trí bắt đầu của RDET là $S_B + N_F * S_F$.

Mỗi entry có 32 byte nên mỗi sector sẽ chứa được đúng 16 entry, vì vậy số entry của RDET thường là bội của 16. Hiện nay các vol hầu như đều có số entry của RDET là 512 (ngoại trừ đĩa 1.44MB là 224 và đĩa 720KB là 112), con số này được chương trình Format lưu vào 2 byte tại offset 11h của Boot Sector.

4.4.4.3 SDET:

Mỗi thư mục trên DOS được lưu trữ giống như một tập tin bình thường. Nội dung của tập tin thư mục này là một dãy entry, mỗi entry chứa tên & thuộc tính của những tập tin và thư mục con giống y như các entry trên RDET. Trên vol có đúng một RDET nhưng có thể có rất nhiều SDET, và cũng có thể không có bảng SDET nào.

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, khi này kích thước bảng SDET sẽ được hệ thống cho tăng thêm bằng cách tìm cluster trống trên vùng DATA và lưu nội dung các entry phát sinh thêm vào cluster mới này (và phần còn lại của cluster nếu có cũng được lưu các entry trống như trên cluster đầu tiên).

4.4.5 Tổ chức bảng thư mục của Windows

4.4.5.1 Cấu trúc entry LFN (Long File Name):

Do DOS đã là HĐH được dùng rộng rãi nhất trên toàn thế giới, nên khi tạo ra Windows công ty Microsoft phải thiết kế sao cho tương thích với DOS càng nhiều càng tốt. Chính vì vậy nên dù phải bổ sung thêm nhiều thuộc tính, và phần tên tập tin phải không bị giới hạn tối đa 8 ký tự như DOS, entry trên Windows vẫn phải có 32 byte giống như entry của 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.

Vì tên tập tin trên Windows có thể lên đến 254 ký tự, và lưu bằng phông chữ UniCode (mỗi ký tự mất 2 byte) nên entry 32 byte chắc chắn không đủ để biểu diễn. Windows giải quyết 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 hai loại entry thuộc tập tin: entry chính chứa tên ngắn và các thuộc tính, entry phụ chứa tên dài tương ứng. Mỗi tập tin sẽ luôn có đúng 1 entry chính, và có 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: entry 0 (32 byte đều giá trị 0) là entry trống chưa hề sử dụng, entry E5 (byte đầu mang giá trị E5h) là entry chính hoặc entry phụ của tập tin đã bị xóa.

Khi Windows tìm entry trống để cấp sẽ tìm entry 0 trước, nếu không có mới tìm entry E5 (để tăng khả năng phục hồi tập tin bị xóa).

4.4.5.1.1 Cấu trúc entry chính:

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

- Phần tên ngắn của entry chính được Windows rút ra 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 du: 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.

4.4.5.1.2 Cấu trúc entry phụ:

Offset	Số byte	Ý nghĩa
0	1	Thứ tự của entry (bắt đầu từ 1)
1	A (10d)	5 ký tự UniCode – bảng mã UTF16

B (11d)	1	Dấu hiệu nhận biết (luôn là 0Fh)
E (14d)	C (12d)	6 ký tự kế tiếp
1C (28d)	4	2 ký tự kế tiếp

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. 01001 010000 011000 0101101

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	

4.4.5.2 RDET:

Với kiến trúc FAT (FAT12/FAT16) thì RDET được tổ chức ở cuối vùng SYSTEM giống như DOS. Nhưng với FAT32, Windows không tổ chức RDET trên SYSTEM, mà coi RDET như SDET – và lưu trên vùng DATA (cluster bắt đầu của RDET được lưu bằng số nguyên 4 byte ở offset 2Ch trên Boot sector).

4.5 Boot Sector

4.5.1 Khái niệm

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 và các thông số quan trọng của vol: kích thước vol, kích thước cluster, kích thước bảng quản lý cluster, ...

4.5.2 Lý do phát sinh

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.

4.5.3 Hình thức tổ chức

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.

4.5.4 Tổ chức thông tin trong BootSector của một số hệ thống tập tin thông dụng

4.5.4.1 Boot Sector của hệ thống FAT (FAT12 & FAT16)

Offset (hex)	Số byte	Ý nghĩa
0	3	Lệnh nhảy đến đầu đoạn mã Boot (qua khỏi vùng thông số)
3	8	Tên công ty /version của HĐH
B	2	Số byte của sector, thường là 512
D	**1**	**Số sector của cluster (S_C)**
E	**2**	**Số sector trước bảng FAT (S_B)**
10	**1**	**Số lượng bảng FAT (N_F)**, thường là 2
11	**2**	**Số Entry của RDET (S_R)**, thường là 512 với FAT16
13	**2**	**Số sector của volume (S_V)**, bằng 0 nếu $S_V > 65535$
15	1	Kí hiệu loại volume
16	**2**	**Số sector của FAT (S_F)**
18	2	Số sector của track
1A	2	Số lượng đầu đọc (side)
1C	4	Khoảng cách từ nơi mô tả vol đến đầu vol
20	**4**	**Kích thước volume (nếu số 2 byte tại offset 13h là 0)**
24	1	Ký hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
25	1	Dành riêng

26	1	Ký hiệu nhận diện HĐH
27	4	SerialNumber của Volume
2B	B	Volume Label
36	8	Loại FAT, là chuỗi "FAT12" hoặc "FAT16"
3E	1CF	Đoạn chương trình Boot nạp tiếp HĐH khi khởi động máy
1FE	2	Dấu hiệu kết thúc BootSector /Master Boot (luôn là AA55h)

Ví dụ, với vol dạng FAT16 có 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 13 đều là 00 nên kích thước vol được lấy ở 4 byte tại offset 20)

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

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

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

4.5.4.2 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 1FE & 1FF) và luôn là 55h, AAh

Ngoài Boot Sector, 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 FAT32:

Offset	Số byte	Nội dung
0	3	Jump_Code: lệnh nhảy qua vùng thông số (như FAT)
3	8	OEM_ID: nơi sản xuất – version, thường là "MSWIN4.1"
B	2	Số byte trên Sector, thường là 512 (như FAT)
D	**1**	**S_C: số sector trên cluster (như FAT)**
E	**2**	**S_B: số sector thuộc vùng Bootsector (như FAT)**
10	**1**	**N_F: số bảng FAT, thường là 2 (như FAT)**
11	2	Không dùng, thường là 0 (số entry của RDET – với FAT)
13	2	Không dùng, thường là 0 (số sector của vol – với FAT)
15	1	Loại thiết bị (F8h nếu là đĩa cứng - như FAT)
16	2	Không dùng, thường là 0 (số sector của bảng FAT – với FAT)
18	2	Số sector của track (như FAT)
1A	2	Số lượng đầu đọc (như FAT)
1C	4	Khoảng cách từ nơi mô tả vol đến đầu vol (như FAT)
20	**4**	**S_V: Kích thước volume** (như FAT)
24	**4**	**S_F: Kích thước mỗi bảng FAT**
28	2	bit 8 bật: chỉ ghi vào bảng FAT active (có chỉ số là 4 bit đầu)
2A	2	Version của FAT32 trên vol này
2C	**4**	**Cluster bắt đầu của RDET**
30	**2**	**Sector chứa thông tin phụ (về cluster trống), thường là 1**
32	**2**	**Sector chứa bản lưu của Boot Sector**
34	C	Dành riêng (cho các phiên bản sau)
40	1	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Dành riêng
42	1	Kí hiệu nhận diện HĐH
43	4	SerialNumber của Volume
47	B	Volume Label
52	**8**	**Loại FAT, là chuỗi "FAT32"**
5A	1A4	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	2	Dấu hiệu kết thúc BootSector /Master Boot (luôn là AA55h)

Tổ chức trong sector chứa thông tin hệ thống file 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 của 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 nguyên 2 byte tại offset 32h> =0006h = 6

4.5.4.3 Boot Sector của hệ thống NTFS

Boot Sector của hệ thống NTFS (New Technology File System) cũng có 6 phần giống như hệ thống FAT & FAT32:

Offset (hex)	Số byte	Nội dung
0	3	Lệnh nhảy
3	8	OEM ID
B	25	BPB
24	48	BPB mở rộng
54	426	Bootrap
1FE	2	Dấu hiệu kết thúc (AA55h)

(Tổ chức của vol NTFS gồm 4 thành phần: Boot Sector, Master File Table (MFT), File System Data, Master File Table Copy).

Sau đây là tổ chức các thông tin quan trọng trong BPB & BPB mở rộng của BootSector NTFS:

Offset	Số byte	Nội dung

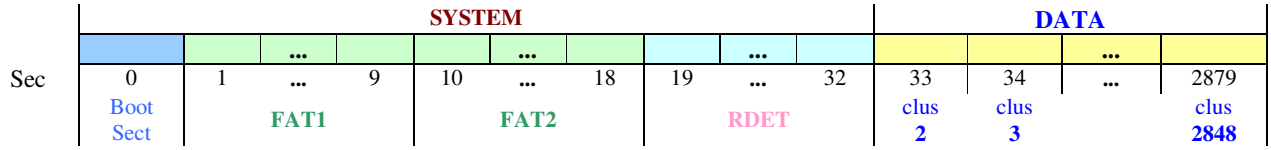
(hex)		
B	2	Số byte của Sector (thường là 512)
D	2	Số sector của Cluster
15	1	Loại đĩa (thường là F8 - đĩa cứng)
28	8	Số sector của Volume
30	8	Vị trí MFT – tính theo chỉ số cluster vật lý
38	8	Vị trí bản sao MFT (chỉ số cluster vật lý)
40	1	Số cluster của MFT record (=2^{-K} nếu K âm)
44	1	Số cluster của Index Buffer
48	8	Serial Number

4.6 Tổng kết

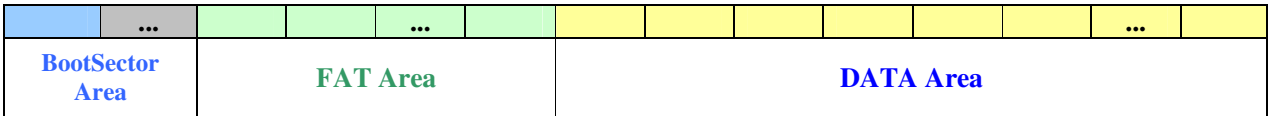
Mô hình tổ chức tập tin trên vol file của HĐH bao gồm các thành phần chính: vùng dữ liệu chứa nội dung tập tin, thường được tổ chức dưới đơn vị khối (cluster) gồm nhiều sector để hệ thống hoạt động tối ưu; vùng hệ thống chứa các thông tin quản lý, bao gồm bảng quản lý cluster để quản lý các khối trên, bảng thư mục để quản lý các tập tin & các thuộc tính liên quan, và một số sector chứa các thông tin quản lý các vùng trên.

Các vol dạng FAT có các thành phần được tổ chức theo thứ tự như sau:

Cụ thể, đĩa mềm 1.44MB thường có cấu trúc:



Các vol được tổ chức theo dạng FAT32 có thứ tự các thành phần như sau:



Trên thực tế có rất nhiều HĐH khác nhau (ngoài các HĐH trên PC mà còn có những HĐH trong máy ảnh số, máy thu âm, điện thoại di động,...). Mỗi HĐH có thể tổ chức hệ thống tập tin trên vol theo những kiến trúc khác nhau, nhưng với những thiết bị lưu trữ có kích thước không lớn thì quản lý cluster theo phương pháp xâu kết hợp chỉ mục là thích hợp. Và để tương thích nhau đa số HĐH đều có cấu trúc vol được thiết kế theo dạng FAT như trên (cụ thể là đĩa mềm, đĩa cứng bỏ túi & các loại Flash Memory).

5. CÀI ĐẶT CÁC CHỨC NĂNG TRÊN VOLUME

5.1 Khái niệm

Để có thể lưu trữ được thông tin /dữ liệu vào hệ thống lưu trữ & truy xuất, sử dụng được các dữ liệu trên hệ thống lưu trữ một cách hiệu quả thì cần phải tuần tự thực hiện các công đoạn:

- Đưa ra khái niệm tập tin, thư mục, volume.
- Xây dựng mô hình thuộc tính & chức năng trên tập tin & thư mục.
- Tổ chức được hình thức lưu trữ tập tin /thư mục & các hình thức quản lý cần thiết khác trên vol.
- Viết thuật giải & chương trình thực hiện các chức năng cần thiết với các tập tin /thư mục trên vol.

Công đoạn thứ tư có thể nói ngắn gọn là "cài đặt các chức năng trên vol", những chức năng cơ bản phải có trước tiên là: định dạng vol, chép tập tin vào vol, liệt kê danh sách tập tin, đọc nội dung tập tin, xóa tập tin.

5.2 Định dạng Volume

5.2.1 Khái niệm

Để vol có thể sử dụng được thì thao tác đầu tiên phải tiến hành chính là định dạng (format) vol. Chức năng này có thể do người sử dụng thực hiện, cũng có thể do nhà sản xuất hoặc người phân phối làm giùm tùy theo loại thiết bị. Bởi vì khi chưa thi hành chức năng định dạng thì vol chỉ là một dãy sector có nội dung rác (những giá trị ngẫu nhiên không đúng với những giá trị cần thiết theo qui định), do đó không thể thực hiện được các thao tác truy xuất tập tin trên vol vì không biết trên vol đang có những tập tin nào, nằm tại đâu, chỗ nào còn trống, kích thước cluster là bao nhiêu,...

Như vậy việc định dạng vol chính là xác định các thông số của từng thành phần trên vol (vị trí, kích thước của cluster, bảng quản lý cluster, bảng thư mục,..) và đưa các giá trị thích hợp vào những thành phần đó. HĐH phải căn cứ vào những thông số này mới "hiểu" được tổ chức tập tin trên vol, từ đó mới thực hiện được các chức năng chép, xóa, xem, sửa, ...

Muốn có một vol trắng để có thể sử dụng bình thường thì sau khi xác định vị trí & kích thước của các thành phần quản lý, ta cần phải lưu các thông số quan trọng đó vào BootSector, sau đó lưu vào các entry trên bảng thư mục các giá trị tương ứng với trạng thái trống, các phần tử trên bảng quản lý cluster cũng vậy – trừ các phần tử tương ứng với các cluster bị hư (nếu hệ thống có quản lý đến trạng thái cluster hư).

Vấn đề có vẻ lớn nhất trong chức năng định dạng chính là việc xác định kích thước bảng quản lý cluster. Khi thực hiện thao tác format thì ban đầu chỉ có kích thước vol & kích thước sector - các thông số còn lại phải tự xác định. Các thông số khác có thể tự phán quyết không được chính xác lắm cũng không gây ảnh hưởng lớn, nhưng kích thước của bảng quản lý cluster phải được tính chính xác.

Với một vol đã được định dạng ta cũng có thể định dạng lại, khi này có 2 trường hợp:

- Định dạng lại hoàn toàn (full format): để tạo ra những dạng thức mới phù hợp hơn cho vol, các thông số của từng thành phần trên vol sẽ được xác định lại. Chức năng này dĩ nhiên cũng được dùng cho những vol chưa được định dạng.
- Định dạng nhanh (quick format): chấp nhận giữ lại các thông số cũ của vol, chỉ cập nhật lại trạng thái các cluster đang chứa dữ liệu thành trống và cho tất cả entry trên bảng thư mục gốc về trạng thái trống. Chức năng này tương đương với việc xóa tất cả mọi tập tin & thư mục đang tồn tại trên vol, nhưng thời gian thi hành rất nhanh – có thể nhanh hơn thời gian xóa một tập tin!

5.2.2 Định dạng cho volume (full format)

5.2.2.1 Thuật giải tổng quát:

(*Đầu vào*: tên vol, kích thước vol, kích thước sector, 2 hàm đọc /ghi sector; *Đầu ra*: vol trắng có nội dung BootSector, bảng quản lý clustor, bảng thư mục hợp lý)

- Bước 1: Xác định giá trị hợp lý cho các thông số cần thiết – ngoại trừ kích thước bảng quản lý cluster.
- Bước 2: Tính kích thước bảng quản lý cluster dựa vào các thông số đã biết.
- Bước 3: Lưu giá trị các thông số trên vào các offset đã qui ước trên BootSector.
- Bước 4: Lưu giá trị tương ứng với trạng thái trống vào các entry trên bảng thư mục.
- Bước 5: Khảo sát các cluster trên vùng dữ liệu để xác định những cluster hư.
- Bước 6: Lưu giá trị tương ứng với trạng thái trống /hư vào các phần tử trên bảng quản lý cluster

5.2.2.2 Định dạng vol theo kiến trúc FAT12 /FAT16

- *Bước 1*: Từ kích thước vol (S_V) và những tiêu chí đặt ra phán quyết giá trị thích hợp cho kích thước cluster (S_C), kích thước RDET (S_R), số bảng FAT (n_F), số sector của vùng BootSector (S_B - số sector trước FAT).

- *Bước 2*: Tính kích thước bảng FAT (S_F) và loại FAT bằng hình thức thử & sai dựa trên đẳng thức $\mathbf{S_B + n_F*S_F + S_R + S_D = S_V}$

(giả sử $\mathbf{S_F = 1}$, từ đó suy ra S_D và kiểm chứng lại xem 2 thông số này có phù hợp nhau không, nếu không thì thử $\mathbf{S_F = 2, 3, 4, ...}$ cho đến khi hợp lý)

- *Bước 3*: Lưu giá trị các thông số trên vào các offset từ 3 – 36h theo đúng vị trí & kích thước đã mô tả trong bảng tham số Vol.

- *Bước 4*: Tạo 1 vùng đệm có kích thước (S_R * 512) byte mang toàn giá trị 0 và lưu vào S_R sector bắt đầu tại sector ($S_B + n_F*S_F$)

- *Bước 5*: Tạo một nội dung đặc biệt rồi ghi xuống & đọc lên từng cluster từ cluster 2 đến cluster [(S_{D+} S_C-1)/S_C]. Nếu không thành công trong việc đọc /ghi, hoặc thời gian đọc /ghi quá lâu, hoặc nội dung đọc được không giống nội dung ghi thì gán cho phần tử quản lý tương ứng trên bảng FAT giá trị FF7 hoặc FFF7 (cluster hư), ngược lại gán 0.

- *Bước 6*: Tạo 1 vùng đệm có kích thước (S_F * 512) byte & đưa các giá trị của bảng FAT vào vùng đệm rồi lưu vào S_F sector bắt đầu tại sector S_B. Nếu $n_F > 1$ thì lưu tiếp vào các vị trí $S_B + S_F$, $S_B + 2*S_F$,..., $S_B + (n_F-1)*S_F$.

*** Ví dụ về việc xác định kích thước bảng FAT:**

Xét đĩa mềm 1.44MB (có 2880 sector), để các tập tin trên vol có thể truy xuất nhanh & an toàn hơn ta có thể cho $S_C = 4$ (sector), $S_B = 1$ (sector), $S_R = 32$ (entry) = 2 (sector), $n_F = 2$.

Thay các giá trị trên vào đẳng thức $S_B + n_F*S_F + S_R + S_D = S_V$ ta được

$1 + 2S_F + 2 + S_D = 2880$ (sector), hay $2S_F + S_D = 2877$ (sector) (*)

(*) $\Rightarrow S_D < 2877$ (sector) = 719.25 (cluster) (vì $S_C = 4$ sector).

$\Rightarrow$ Loại FAT tối ưu nhất (về kích thước) là **FAT12**, vì $S_D < \mathbf{4079}$ (cluster)

* Giả sử $S_F = 1$ (sector): (*) $\Rightarrow$ $S_D =$ 2875 (sector) = 718.75 (cluster)

$\Rightarrow$ Vùng dữ liệu có 718 cluster, nên bảng FAT phải có 718 + 2 = 720 phần tử, do đó $S_F = (720*1.5)/512 =$ 2.1x (sector)

⇒ Bảng FAT phải chiếm 3 sector – mâu thuẫn với giả thiết $S_F = 1$**.** Vậy kích thước bảng FAT của vol này không thể là 1 sector

* Giả sử $S_F = 2$ (sector): tương tự, ta vẫn thấy mâu thuẫn, tức kích thước bảng FAT phải lớn hơn 2 sector.

* Giả sử $S_F = 3$ (sector): (*) ⇒ $S_D = 2871$ (sector) = 717.75 (cluster).

⇒ Vùng dữ liệu có 717 cluster, nên bảng FAT phải có 717 + 2 = 719 phần tử, do đó $S_F = (719*1.5)/512 =$ 2.1x (sector)

⇒ Bảng FAT phải chiếm 3 sector – phù hợp với giả thiết SF = 3. Vậy kích thước bảng FAT của vol này là 3 sector.

5.2.2.3 Định dạng vol theo kiến trúc FAT32

Thực hiện tương tự, nhưng không cần xác định kích thước RDET và loại FAT, bản RDET sẽ nằm tại cluster trống đầu tiên (cluster 2 nếu nó không bị hư) & có kích thước bằng kích thước cluster. Ngoài ra có thêm bước lưu trữ bản backup của Boot Sector và thông tin về vùng trống trên Sector chứa thông tin hệ thống.tập tin (nếu không có cluster hư: số cluster trống = tổng số cluster-1, chỉ số cluster trống bắt đầu là 3).

5.2.3 Định dạng nhanh (quick format)

5.2.3.1 Thuật giải tổng quát:

(*Đầu vào*: tên vol cần format lại; *Đầu ra*: vol trắng có thông số các thành phần quản lý không thay đổi)

- Bước 1: Đọc BootSector để xác định các thông số cần thiết.
- Bước 2: Đọc bảng quản lý cluster vào bộ nhớ.
- Bước 3: Giữ nguyên danh sách các cluster hư (nếu có) & cho các trạng thái các cluster còn lại thành trống.
- Bước 4: Lưu lại bảng quản lý cluster vào vol.
- Bước 5: Tạo 1 vùng đệm có kích thước bằng kích thước bảng thư mục gốc & đưa vào vùng đệm các giá trị tương ứng với các entry trống.
- Bước 6: Ghi vùng đệm trên vào vị trí lưu trữ bảng thư mục gốc của vol.

5.2.3.2 Định dạng nhanh vol theo kiến trúc FAT12 /FAT16

- *Bước 1*: Đọc BootSector để xác định các thông số S_B, S_F, n_F, S_R.

- *Bước 2*: Đọc bảng FAT vào mảng aFAT trên bộ nhớ.

- *Bước 3*: Gán giá trị FREE (giá trị 0) vào các phần tử có giá trị khác BAD (FF7 với FAT12 hoặc FFF7 với FAT16) trên mảng aFAT.

- *Bước 4*: Lưu lại mảng aFAT trên vào n_F bảng FAT trên vol (*lưu vào S_F sector bắt đầu tại sector S_B, nếu $n_F > 1$ thì lưu tiếp vào các vị trí $S_B + S_F$, $S_B + 2*S_F$,..., $S_B + (n_F-1)*S_F$*)

- *Bước 5*: Tạo vùng đệm S_R (byte) có nội dung các byte đều là 0.

- *Bước 6*: Lưu lại vùng đệm trên vào bảng RDET trên vol (*lưu vào S_R sector bắt đầu tại sector ($S_B + n_F*S_F$)*)

5.3 Đọc nội dung tập tin trên Volume

5.3.1 Khái niệm

Đây là thao tác truy xuất vol được thực hiện nhiều nhất, cũng là thao tác thường xuyên của hệ thống máy tính. Mà tốc độ truy xuất bộ nhớ ngoài (nơi chứa tập tin) chậm hơn nhiều so với bộ nhớ trong, do đó để tăng tốc độ hoạt động của máy tính – đồng thời để có thể bảo mật dữ liệu, kiểm chứng sự hợp lý, đề kháng với các sự cố có thể gây hư hỏng, ... – khi truy xuất tập tin người ta thường dùng tới nhiều hệ thống Cache, nhiều kỹ thuật tối ưu & các xử lý khác. Tuy nhiên để có thể dễ dàng hơn cho việc nắm được một cách cơ bản tổ chức lưu trữ tập tin & cơ chế hoạt động của hệ thống quản lý tập tin, các thuật giải được trình bày sau đây chỉ nêu cách giải quyết cơ bản, chân phương nhất - không quan tâm nhiều đến việc tối ưu & các xử lý nâng cao khác.

Ta đã biết về cơ bản có hai loại tập tin khác nhau: tập tin bình thường và tập tin thư mục (còn gọi là thư mục con). Cho nên trước mắt có thể thấy có 2 thao tác khác nhau: đọc nội dung tập tin bình thường & đọc nội dung bảng thư mục con (tức nội dung tập tin thư mục - SDET). Nhưng tập tin bình thường và thư mục đều có thể nằm trong một SDET nào đó, vì vậy có thể phân ra tới 4 thao tác tương đối riêng biệt: đọc nội dung một tập tin bình thường ở RDET, đọc nội dung một thư mục con (SDET) ở RDET, đọc nội dung một tập tin bình thường ở SDET, và đọc nội dung một thư mục con nằm trong một SDET nào đó của vol.

5.3.2 Đọc nội dung tập tin ở RDET của vol dạng FAT:

- *Bước 1*: Đọc BootSector để xác định các thông số S_B, S_F, S_R, S_C, S_S, N_F.

- *Bước 2*: Đọc bảng RDET vào bộ nhớ.

- *Bước 3*: Tìm trên RDET vừa đọc entry chính tương ứng với tập tin cần lấy nội dung (*nếu không có thì thông báo tập tin không tồn tại & thoát*).

- *Bước 4*: Đọc bảng FAT vào mảng aFAT trên bộ nhớ.

- *Bước 5*: Từ chỉ số cluster bắt đầu $\mathbf{f_1}$ trong entry tìm được ở bước 3 xác định dãy các phần tử kế tiếp $\mathbf{f_2, f_3, f_4, ...}$ theo công thức $\mathbf{f_{i+1} = aFAT\ [\ f_i\ ]}$ cho đến khi gặp chỉ số $\mathbf{f_N}$ có $\mathbf{aFAT[\ f_N\ ] = EOF.}$

- *Bước 6*: Đọc những cluster $\mathbf{f_1, f_2, f_3, ..., f_N}$ & ghép các nội dung đọc được lại theo đúng thứ tự đó ta được nội dung tập tin (*riêng phần nội dung trên cluster* $\mathbf{f_N}$ *chỉ lấy M byte, với M =* <*kích thước tập tin*> *MOD* $\boldsymbol{S_C}$).

**** Thuật giải chi tiết cho việc tìm entry chính ứng với tập tin ở Bước 3:***

- Bước 3.1: Chuyển tên tập tin về dạng qui ước trên entry chính (8 ký tự tên chính nối với 3 ký tên mở rộng, nếu không đủ thì trám ký tự trắng vào, các ký tự chữ cái thường đổi sang hoa,..). Nếu không thể chuyển được thì qua Bước 3.3.
- Bước 3.2: Tìm trên bảng thư mục entry có phần tên giống với chuỗi vừa chuyển. Nếu tìm được thì ghi nhận chỉ số entry K và nhảy đến bước 3.9.
- Bước 3.3: K = 0.
- Bước 3.4: Nếu K= Tổng số entry của bảng thư mục hoặc byte đầu của entry K là 0 thì gán K=-1 và nhảy đến bước 3.9.
- Bước 3.5: Kiểm tra entry K có phải là entry chính (offset 0 khác E5h, offset Bh khác 0Fh), nếu không phải thì nhảy đến bước 3.8.
- Bước 3.6: Xác định tên file dài từ các entry phụ của entry chính K.
- Bước 3.7: Nếu tên file dài vừa xác định giống với tên tập tin thì nhảy đến bước 3.9.
- Bước 3.8: K = K+1, quay lại bước 3.4.
- Bước 3.9: Kết thúc, K chính là chỉ số của entry cần tìm (nếu không tìm được thì K=-1)

5.3.3 Đọc nội dung SDET thuộc RDET của vol FAT:

Việc sử dụng nội dung tập tin bình thường và nội dung của tập tin thư mục là hoàn toàn khác nhau. Nhưng nếu tổ chức lưu trữ tập tin thư mục giống như tập tin bình thường thì việc đọc nội dung bảng thư mục con cũng sẽ giống như đọc nội dung tập tin bình thường. Với vol được tổ chức theo cấu trúc FAT12 /FAT16 thì phần nội dung tập tin thư mục trên cluster cuối cùng được sử dụng hết chứ không như tập tin bình thường có thể chỉ chiếm một phần. (vì vậy trường <kích thước tập tin> trong entry của tập tin thư mục sẽ không được sử dụng – HĐH luôn gán là 0).

5.3.4 Đọc nội dung tập tin trong SDET của vol FAT:

Entry chứa các thông tin về tập tin cần đọc sẽ nằm trong SDET của thư mục chứa tập tin đó. Có thể thấy một vấn đề trước tiên là thư mục con chứa tập tin có thể đang nằm trong một thư mục con khác, và thư mục con này lại có thể nằm trong một thư mục con khác nữa. Như vậy để đọc được nội dung tập tin trước tiên ta phải xác định đường dẫn đến nơi chứa tập tin (là một dãy các thư mục con mà thư mục trước là cha của thư mục sau), từ đường dẫn này ta sẽ lần lượt đọc & phân tích các bảng thư mục con để cuối cùng có được bảng thư mục con của thư mục chứa tập tin. Giả sử các thư mục trong đường dẫn này được gọi theo thứ tự là thư mục con cấp 1, thư mục con cấp 2,.. thì hình thức để xác định nội dung của SDET cấp N là:

- Tìm entry chính tương ứng với thư mục con cấp 1 trong RDET và suy được nội dung của tập tin này (SDET cấp 1).

- Tìm entry chính ứng với thư mục con cấp 2 trong SDET cấp 1 trên và suy được nội dung của tập tin thư mục con cấp 2 (SDET cấp 2).

- ....

- Tìm entry chính ứng với thư mục con cấp N trong bảng SDET cấp N-1 trên và suy được nội dung của tập tin thư mục con cấp N (SDET cấp N)

Thuật giải:

(*Đầu vào*: tên vol, đường dẫn & tên tập tin cần lấy nội dung; *Đầu ra*: nội dung tập tin)

- Bước 0: Phân tích đường dẫn để xác định số cấp (giả sử là N) & tên của các thư mục con ở các cấp.
- Bước 1: Đọc BootSector của vol để xác định các thông số cần thiết (*vị trí & kích thước của RDET & bảng quản lý cluster, kích thước cluster, vị trí bắt đầu của vùng dữ liệu,...*)
- Bước 2: Đọc RDET vào bộ nhớ.
- Bước 3: Đọc bảng quản lý cluster vào bộ nhớ.
- Bước 4: Bắt đầu từ RDET lần lượt suy ra các SDET ở các cấp kế tiếp & cuối cùng được SDET chứa entry của tập tin (SDET cấp N)
- Bước 5: Tìm trên SDET hiện tại entry chính ứng với tập tin cần lấy nội dung (*nếu không có thì thông báo tập tin không tồn tại & thoát*).
- Bước 6: Từ nội dung bảng quản lý cluster đã đọc ở bước 3 & thông tin của entry tìm được ở bước trên suy ra danh sách những cluster chứa nội dung của tập tin.
- Bước 7: Đọc những cluster tương ứng với danh sách trên & ghép các nội dung đọc được lại theo đúng thứ tự đó ta được nội dung tập tin (*riêng phần nội dung trên cluster cuối chỉ lấy N byte, với N = <kích thước tập tin> MOD <kích thước cluster>*).

**** Thuật giải chi tiết cho Bước 4:***

- Bước 4.1: K = 1.

- Bước 4.2: Tìm trên bảng thư mục hiện tại entry có phần tên giống với tên thư mục con cấp K (*nếu không có thì thông báo đường dẫn sai & thoát*).
- Bước 4.3: Từ nội dung bảng quản lý cluster đã đọc ở bước 3 & thông tin của entry tìm được ở bước trên suy ra danh sách những cluster chứa nội dung tập tin thư mục con cấp K.
- Bước 4.4: Đọc những cluster tương ứng với danh sách trên & ghép các nội dung đọc được lại theo đúng thứ tự đó ta được SDET của thư mục cấp K.
- Bước 4.5: Gán bảng thư mục hiện tại là SDET trên
- Bước 4.6: Nếu K < N thì tăng K thêm 1 và quay lại bước 4.2.

5.3.5 Đọc nội dung thư mục con trong SDET:

Bài toán này đã được giải quyết ở chức năng kế trước (bằng cách thực hiện từ đầu cho đến hết bước 4).

5.4 Lưu giữ tập tin vào Volume

5.4.1 Khái niệm

Sau thao tác đầu tiên là định dạng vol, chức năng kế tiếp có ảnh hưởng đến nội dung lưu trữ trên vol là đưa tập tin vào vol, chức năng này sẽ được thực hiện nhiều lần trong quá trình sử dụng vol. Việc đưa một tập tin vào vol cụ thể là chép một tập tin từ nơi khác vào vol, tạo một tập tin trên vol, hoặc tạo một thư mục con trên vol. Khi vol đang ở trạng thái trống thì các thao tác đó chỉ có thể thực hiện trên thư mục gốc của vol, nhưng khi vol đã có thư mục con thì những thao tác trên có thể thực hiện trong một thư mục con nào đó của vol.

Ta có thể phân ra 4 thao tác tương đối riêng biệt: đưa một tập tin bình thường vào thư mục gốc của vol, tạo một thư mục con ở thư mục gốc của vol, đưa một tập tin bình thường vào một thư mục con nào đó của vol, tạo một thư mục con trong một thư mục con nào đó của vol.

5.4.2 Đưa một tập tin vào thư mục gốc của vol FAT:

- *Bước 1*: Đọc BootSector để xác định các thông số S_B, S_F, n_F, S_R, S_C, S_S.

- *Bước 2*: Đọc bảng RDET vào bộ nhớ.

- *Bước 3*: Xác định số entry phụ cần dùng để lưu tên dài M = (<số ký tự của tên tập tin> +12) / 13.

- *Bước 4*: Tìm trên bảng RDET vừa đọc M+1 entry trống liên tiếp (*nếu không có thì thông báo hết chỗ lưu trữ & thoát*).

- *Bước 5*: Xác định số cluster **N** mà tập tin sẽ chiếm (N=<*kích thước nội dung tập tin*>/S_C).

- *Bước 6*: Đọc bảng FAT vào mảng aFAT trên bộ nhớ.

- *Bước 7*: Tìm trên mảng aFAT N phần tử $\mathbf{f_1, f_2, ..., f_N}$ có giá trị 0 (*nếu không có đủ N phần tử thì thông báo không đủ dung lượng trống để lưu nội dung tập tin & thoát*).

- *Bước 8*: Điều chỉnh lại giá trị N phần tử trên theo công thức: aFAT [f_i] = f_{i+1}, với i từ 1 đến (n-1), và aFAT [f_N] = EOF.

- *Bước 9*: Đưa vào entry chính tìm được ở bước 4 tên chính & các thuộc tính của tập tin (trường cluster bắt đầu được gán giá trị f_1)

- *Bước 10*: Đưa vào entry phụ ở bước 4 tên dài & các thuộc tính cần thiết.

- *Bước 11*: Lưu lại các entry vừa điều chỉnh vào RDET của vol.

- *Bước 12*: Lưu lại mảng aFAT trên vào n_F bảng FAT trên vol.

- *Bước 13*: Phân vùng nội dung tập tin ra thành những đoạn có kích thước bằng S_C và lưu chúng vào các cluster tương ứng mang chỉ số f_1, f_2, ..., f_N trên vùng DATA.

5.4.3 Tạo thư mục con ở thư mục gốc của vol FAT

Tương tự thuật giải trên, nhưng chỉ cần tìm 1 cluster trống cho nội dung tập tin thư mục và lưu vào cluster này nội dung của bảng thư mục con tương ứng với một thư mục rỗng (2 entry đầu có tên là "." và "..", các entry còn lại đều mang giá trị 0.

5.4.4 Đưa tập tin vào thư mục con của vol FAT

Phân tích đường dẫn để xác định số cấp N & tên các thư mục con ở các cấp. Từ đó đọc bảng SDET cấp N chứa entry của tập tin cần đưa vào (xem thuật toán đọc SDET đã trình bày ở phần 5.3). Nếu *số entry trống trên SDET không đủ đáp ứng thì nối thêm một cluster trống vào*, và sau đó thực hiện các thao tác tương tự như đưa tập tin vào RDET đã trình bày phía trên.

5.4.5 Tạo một thư mục trong một thư mục con có sẵn trên vol dạng FAT

(Đầu vào: tên vol, đường dẫn đến nơi thư mục con có sẵn, tên & các thuộc tính của thư mục con cần tạo; *Đầu ra*: một thư mục rỗng được tạo ra trong thư mục con đã chỉ ra theo đúng cấu trúc lưu trữ đã thiết kế - không ảnh hưởng đến các nội dung đang có sẵn trên vol)

Cũng tương tự như các thao tác trên, ta đọc bảng SDET cha của thư mục cần tạo, và nếu số entry trống trên SDET này không đủ đáp ứng thì *nối thêm một cluster trống vào*, và sau đó thực hiện các thao tác tương tự như tạo thư mục trong RDET đã trình bày phía trên.

5.5 Xóa tập tin

5.5.1 Khái niệm

Ta có nhiều dạng xóa tập tin: xóa bình thường (có thể phục hồi lại được – nhưng khả năng thành công càng thấp dần theo thời gian), xóa chắn chắn phục hồi được, xóa sao cho không thể phục hồi được, xóa thư mục, xóa các tập tin rác,…

5.5.2 Xóa tập tin bình thường

Tương tự như thao tác đọc nội dung tập tin, ta cần xác định được entry chính và các entry phụ ứng với tập tin, và suy ra được các cluster chứa nội dung tập tin. Nhưng thay vì đọc các cluster này lên để được nội dung tập tin thì ta cần xóa để chúng thành các cluster trống, và đồng thời xóa luôn các entry ứng với tập tin (bằng cách gán byte đầu của entry thành E5h, 31 byte còn lại của entry giữ nguyên).

Việc phục hồi lại tập tin đã xóa theo cách này có thể thực hiện được vì việc tìm lại entry tương ứng với tập tin đã xóa thường sẽ thành công. Do khi cấp entry trống Windows ưu tiên sử dụng các entry 0 (chỉ khi không còn entry 0 mới dùng đến entry E5) nên các entry E5 hầu như không bị ảnh hưởng. Thay giá trị E5 thành giá trị khác là ta đã phục hồi được các thuộc tính của tập tin, từ số lượng cluster mà tập tin chiếm ta tìm đúng bấy nhiêu đó cluster trống kể từ chỉ số cluster bắt đầu của tập tin – và phán luôn đó chính là những cluster thuộc tập tin. Dĩ nhiên như vậy chưa chắc đúng nhưng khả năng đúng cũng khá cao!

Trên HĐH DOS chỉ có chức năng xóa này, còn trên Windows muốn thực hiện chức năng này ta phải đè phím <Shift> và ấn phím <Del> (hoặc đè <Shift> và chọn "Delete" trên menu)

5.5.3 Xóa tập tin cho mất hẳn

Thực hiện cũng giống như trên, nhưng thay vì chỉ gán E5h vào byte đầu của entry và giữ nguyên 31 byte còn lại thì ta đưa nội dung rác vào 31 byte này, đồng thời ghi đè nội dung rác vào các cluster chứa nội dung tập tin.

Với những nơi có khả năng chế tạo được thiết bị lưu trữ thì cũng có khi chế tạo được các thiết bị có thể đọc được những dữ liệu đã bị ghi đè lên vài lần (chi phí cho việc này rất tốn kém và chỉ những thông tin cực kỳ quý giá người ta mới phục hồi bằng hình thức này). Để ngăn chặn việc phục hồi theo cách này ta chỉ cần đơn giản ghi đè khá nhiều lần (7 lần là đủ!)

5.5.4 Xóa tập tin sao cho phục hồi được

Vấn đề trước mắt là trạng thái những cluster chứa nội dung tập tin sẽ như thế nào sau khi xóa. Nếu chuyển thành cluster trống thì đến một lúc nào đó hệ thống sẽ dùng để lưu nội dung các tập tin khác và không còn phục hồi được nữa, nhưng nếu không chuyển thành cluster trống thì sau khi xóa không gian trống trên đĩa vẫn không được tăng thêm – cũng không ổn.

Giải pháp hợp lý là nén tập tin lại cho nhỏ hơn và lưu vào một nơi đặc biệt nào đó và tiến hành xóa tập tin ban đầu theo hình thức xóa bình thường. Như vậy chắc chắn phục hồi lại được và không gian trống cũng được tăng thêm sau khi xóa.

5.5.5 Xóa thư mục

Trước tiên phải xóa tất cả các tập tin bên trong thư mục (gọi đệ qui nếu bên trong thư mục lại có thư mục con), sau đó tiến hành xóa tập tin thư mục như xóa tập tin bình thường.