HỆ ĐIỀU HÀNH

Phạm Đăng Hải
haipd@soict.hut.edu.vn

Bộ môn Khoa học Máy tính
Viện Công nghệ Thông tin & Truyền Thông

Ngày 13 tháng 8 năm 2014

Chương 4 Quản lý hệ thống file

Giới thiệu

- Bộ nhớ ngoài (*đĩa từ, băng từ, đĩa quang,..*): dung lượng lớn và cho phép lưu trữ lâu dài
  - Được người dùng sử dụng lưu trữ dữ liệu và chương trình
  - Dữ liệu và chương trình được lưu dưới dạng file (*tập tin/tệp*) ⇒ Tạo nên hệ thống file
  - Hệ thống file gồm 2 phần riêng biệt
    - Các file: Chứa dữ liệu/chương trình của hệ thống/người dùng
    - Cấu trúc thư mục : Cung cấp các thông tin về file
- Hệ thống file lớn ⇒ Quản lý như thế nào?
  - Các thuộc tính của file, thao tác cần phải cung cấp?
- Lưu trữ và truy xuất dữ liệu trên thiết bị lưu trữ như thế nào?
  - Phương pháp cung cấp không gian lưu trữ, quản lý vùng tự do

⇒Khó khăn phải trong suốt với người dùng (*tính thuận tiện*)

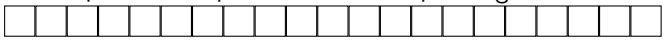
- Các file dữ liệu /chương trình có thể sử dụng chung
  - Đảm bảo tính toàn vẹn dữ liệu và loại bỏ truy nhập bất hợp lệ?
- Dữ liệu không lưu trữ tập trung ⇒ hệ thống file phân tán
  - Truy nhập file từ xa, đảm bảo tính toàn vẹn...

Nội dung chính

Nội dung chính

Giới thiệu

- Thông tin lưu trữ trên nhiều phương tiện/thiết bị lưu trữ khác nhau
  - Ví dụ: Đĩa từ, băng từ, đĩa quang...
  - Thiết bị lưu trữ được mô hình như một mảng của các khối nhớ



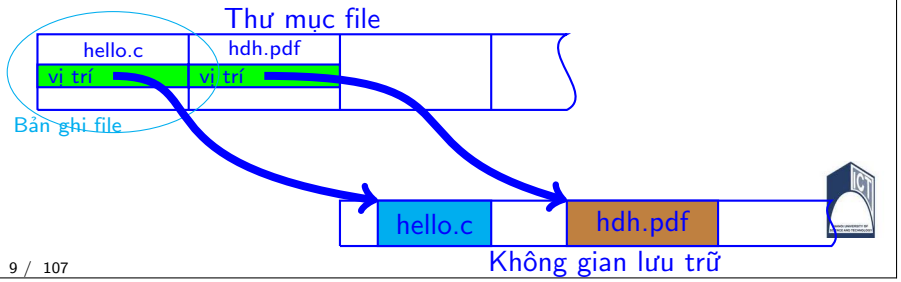
- File là tập thông tin ghi trên thiết bị lưu trữ.
  - File là đơn vị lưu trữ của hệ điều hành trên bộ nhớ ngoài
  - File bao gồm dãy các bits, bytes, dòng, bản ghi,... mang ý nghĩa được định nghĩa bởi người tạo ra
- Cấu trúc của file được định nghĩa theo loại file
  - File văn bản: Chuỗi ký tự tổ chức thành dòng
  - File đối tượng: Bytes được tổ chức thành khối để chương trình liên kết (*linker*) hiểu được
  - File thực thi: Chuỗi các mã lệnh có thể thực hiện trong bộ nhớ
  - ...

Các thuộc tính file

- Tên file (*Name*): Chuỗi ký tự (*hello.c*)
  - Thông tin lưu dưới dạng người dùng có thể đọc được
  - Có thể phân biệt chữ hoa/chữ thường
  - Đảm bảo tính độc lập của file với tiến trình, người dùng...
    - A tạo file *hello.c* bằng *notepad* trên hệ *Windows*
    - B dùng *emacs* trên *linux* sửa lại file bởi xác định tên *hello.c*
- Định danh (*Identifier*): Thẻ xác định duy nhất một file
- Kiểu (*Type*): Dùng cho hệ thống hỗ trợ nhiều kiểu file
  - Có thể xác định kiểu file dựa trên một phần của tên file
    - Ví dụ: .exe, .com/ .doc, .txt/ .c, .jav, .pas/ .pdf, .jpg,...
  - Dựa trên kiểu, HĐH sẽ thao tác trên tập tin phù hợp
    - Thực hiện file thực thi mà file nguồn đã sửa ⇒ Dịch lại
    - Nháy đúp vào một file văn bản (*.doc)⇒ Gọi word processor
- Vị trí (*Position*): Trỏ tới thiết bị và vị trí của file trên đó
- Kích thước (*Size*): Kích thước hiện thời/ tối đa của file
- Bảo vệ (*Protection*): Điều khiển truy nhập: Ai có thể đọc/ghi..
- Thời gian (*Time*): Thời điểm tạo, sửa đổi, sử dụng cuối ...

Các thuộc tính file (tiếp tục)

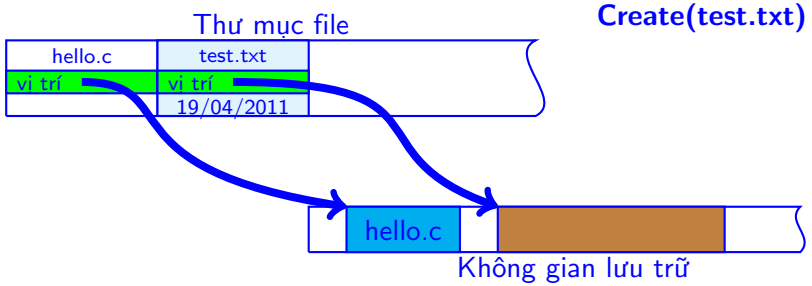
- Thuộc tính file được lưu trong cấu trúc dữ liệu: **Bản ghi file**
  - Có thể chỉ chứa tên file và định danh file; định danh file xác định các thông tin còn lại
  - Kích thước từ vài bytes lên tới kilobytes
- Các bản ghi file được lưu giữ trong **Thư mục file**
  - Kích thước có thể đạt tới Megabytes
  - Thường được lưu trữ trên thiết bị nhớ ngoài
  - Được đưa từng phần vào bộ nhớ khi cần thiết



Các thao tác cơ bản

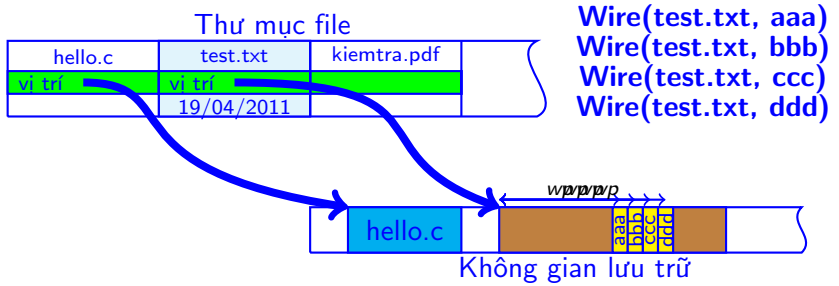
1. Tạo file (*Create*)
2. Ghi file (*Write*)
3. Đọc file (*Read*)
4. Thay đổi vị trí trong file (*Seek*)
5. Xóa file (*Delete*)
6. Thu gọn file (*Truncate*)
7. ...

Các thao tác cơ bản : Tạo file



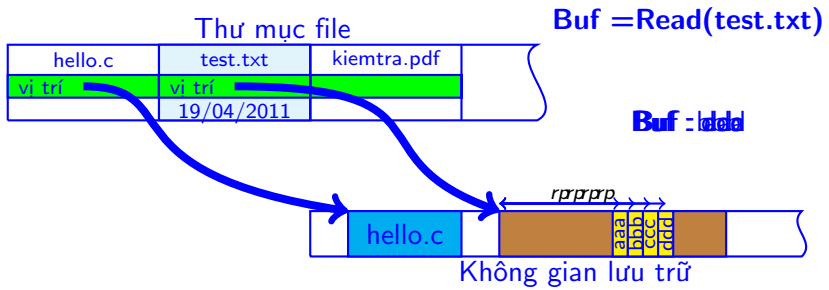
- Tìm vùng tự do trong không gian lưu trữ của hệ thống file
  - Cung cấp vùng trống như thế nào?
- Tạo một phần tử mới trong thư mục file
- Lưu tên file, vị trí của file và các thông tin khác

Các thao tác cơ bản : Ghi file



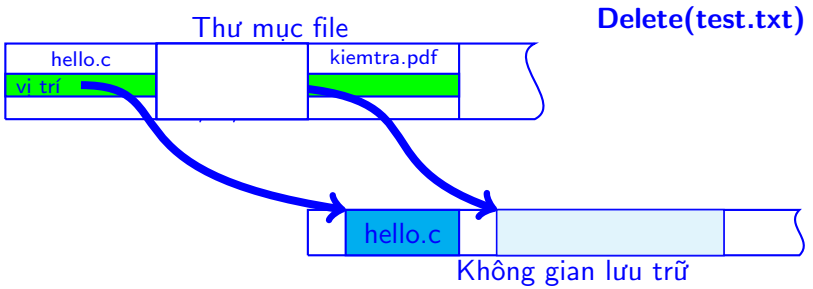
- Lời gọi hệ thống Write() yêu cầu tên file và dữ liệu được ghi
- Dùng tên file, tìm kiếm file trong thư mục file
- Dựa vào trường vị trí, tìm vị trí của file trên thiết bị lưu trữ
- Hệ thống lưu con trỏ ghi (*write pointer*) để chỉ ra vị trí ghi
  - Con trỏ ghi thay đổi sau mỗi thao tác ghi

Các thao tác cơ bản : Đọc file



- Lời gọi hệ thống Read() yêu cầu tên file và vùng đệm ghi KQ
- Dùng tên file, tìm kiếm file trong thư mục file
- Dựa vào trường vị trí, tìm vị trí của file trên thiết bị lưu trữ
- Hệ thống lưu con trỏ đọc (*read pointer*) chỉ ra vị trí được đọc
  - Con trỏ đọc thay đổi sau mỗi thao tác đọc dữ liệu
- Dùng một con trỏ cho cả thao tác đọc và ghi: **con trỏ file**

Các thao tác cơ bản : Xóa file



- Dùng tên file, tìm kiếm file trong thư mục file
- Vùng nhớ được xác định bởi 2 trường vị trí và kích thước được giải phóng để có thể sử dụng lại bởi các file khác
- Xóa phần tử tương ứng trong thư mục file
- Xóa logic / xóa vật lý

Các thao tác cơ bản : Thay đổi vị trí trong file và thu gọn file

- Thay đổi vị trí trong file
  - Duyệt thư mục để tìm phần tử tương ứng
  - Con trỏ file được thay bằng giá trị thích hợp
  - Thao tác này không yêu cầu một hoạt động vào/ra
- Thu gọn file
  - Được sử dụng khi người sử dụng muốn xóa nội dung file nhưng vẫn giữ nguyên các thuộc tính
  - Tìm kiếm file trong thư mục file
  - Đặt kích thước file về 0
  - Giải phóng vùng nhớ dành cho file

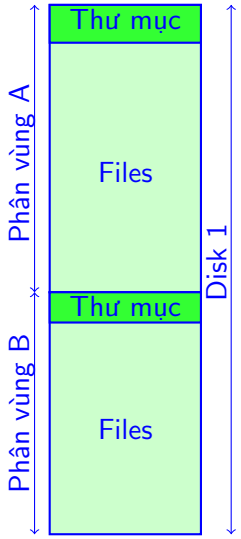
Các thao tác cơ bản : Một số thao tác khác

- Ngoài các thao tác cơ bản, còn tồn tại nhiều thao tác khác
  - Thêm dữ liệu vào cuối file (*append*)
  - Lấy/đặt thông tin thuộc tính file
  - Đổi tên file
- Có thể được đảm bảo thông qua các thao tác cơ bản. Ví dụ copy file
  - Tạo file mới
  - Đọc dữ liệu từ file cũ
  - Ghi ra file mới

Các thao tác cơ bản: Đóng mở file

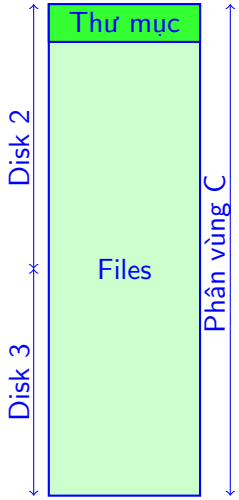
- Các thao tác file phải duyệt thư mục file ⇒ Lãng phí thời gian
- Để giải quyết, các tiến trình phải thực hiện mở file (*open*) trước khi thao tác với file
  - Thao tác mở file: tìm kiếm file trong thư mục file
  - Chép phần tử tương ứng vào bảng file mở
    - Chứa thông tin về các file đang được mở
  - Trả lại con trỏ của phần tử tương ứng trong bản file mở
- Khi có yêu cầu, HĐH tìm kiếm trong bảng file mở
  - Dùng con trỏ trả về của thao tác mở file
- Khi không sử dụng file nữa cần phải đóng (*close*) file.
  - HĐH sẽ loại bỏ phần tử tương ứng trong bảng file mở
- Thao tác đóng/mở file trong môi trường đa người dùng
  - Dùng 2 loại bảng file mở: Cho từng tiến trình và cho hệ thống
  - Ghi lại số tiến trình đang mở file (*File Open Counter*)
    - Tăng/Giảm bộ đếm khi có tiến trình mở/đóng file
    - Xóa p/tử tương ứng trong bảng file mở mức hệ thống khi bộ đếm bằng không

Các phân vùng (Partition)



- Đĩa được chia thành nhiều phân vùng
  - Partitions, Minidisks, Volumes
- Mỗi phân vùng được xử lý như vùng lưu trữ phân biệt
- Có thể chứa một HĐH riêng

Các phân vùng (Partition)



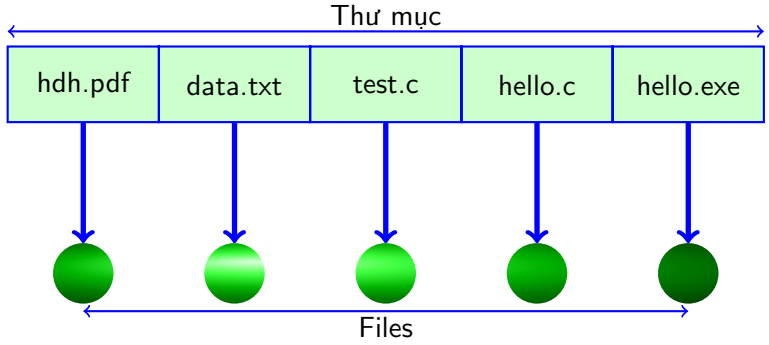
Kết hợp một vài đĩa thành một cấu trúc logic lớn

- Người dùng chỉ quan tâm tới cấu trúc file và thư mục logic
- Không quan tâm tới cách phân phối vật lý không gian đĩa cho files

Các thao tác với thư mục

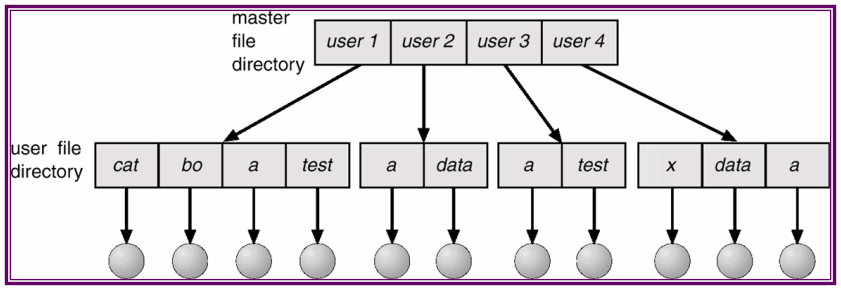
- Mỗi một phân khu lưu các thông tin về file trong nó
  - Các thông tin file được lưu trữ trong *thư mục thiết bị* - thư mục
- Thư mục là bảng chuyển cho phép ánh xạ từ một tên (*file*) thành một phần tử trong thư mục
  - Thư mục có thể được cài đặt bằng nhiều cách khác nhau
    - Yêu cầu các thao tác chèn, tạo mới, xóa, duyệt danh sách
- Các thao tác
  - **Tìm kiếm file:** Tìm phần tử ứng với một file xác định
  - **Tạo file:** Tạo file mới cần tạo phần tử trong thư mục
  - **Xóa file:** Khi xóa file, xóa phần tử tương ứng trong thư mục
  - **Liệt kê thư mục:** Liệt kê files và nội dung phần tử tương ứng trong thư mục
  - **Đổi tên file:** Thay đổi tên file, vị trí trong cấu trúc thư mục
  - **Duyệt hệ thống file:** Truy nhập tất cả thư mục và nội dung tất cả các files trong thư mục (*backup dữ liệu lên băng từ*)

Thư mục một mức



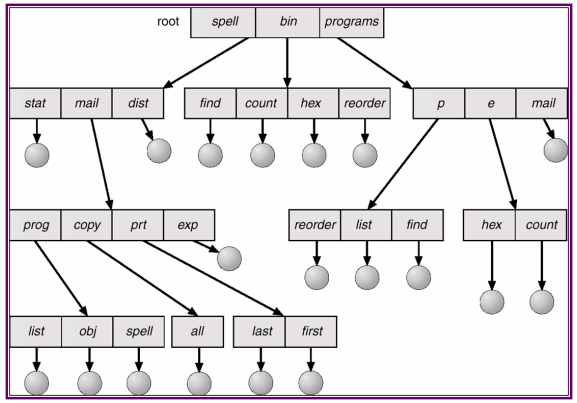
- Cấu trúc đơn giản nhất, các file nằm trong cùng một thư mục
- Số người dùng và số file lớn, khả năng trùng tên file cao
  - Mỗi người dùng một thư mục riêng

Thư mục hai mức



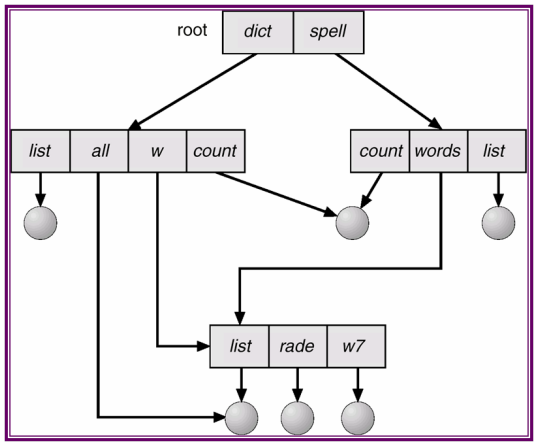
- Mỗi người sử dụng có một thư mục riêng, khi làm việc với file chỉ duyệt thư mục riêng
- Khi log in, hệ thống sẽ kiểm tra và cho phép người sử dụng làm việc với thư mục riêng
- Khi thêm một người dùng
  - Hệ thống tạo phần tử mới trong *Master file directory*
  - Tạo ra *User file directory*
- Giả quyết v/đề trùng tên; Hiệu quả khi người dùng độc lập
- Khó khăn khi muốn dùng chung file

Thư mục cấu trúc cây



- Tồn tại một đường dẫn (*tương đối/tuyệt đối*) đến một file
- Thư mục con là file được xử lý đặc biệt (*bit đánh dấu*)
- Các thao tác tạo/xóa/duyệt... t/hiện trên thư mục hiện thời
  - Xóa thư mục con ⇒ Xóa hết các cây con của nó

Thư mục dùng chung



- Người dùng có thể *link* đến một file của người dùng khác
- Khi duyệt thư mục (*backup*) file có thể duyệt nhiều lần
- Xóa file: liên kết/ nội dung (*người tạo file /liên kết cuối*)

Nội dung chính

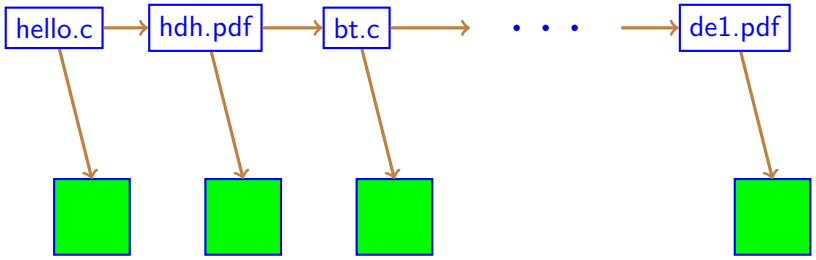
Phương pháp

1. Danh sách tuyến tính với con trỏ tới các khối dữ liệu
   - Đơn giản cho lập trình
   - Tốn thời gian khi thực hiện các thao tác với thư mục
     - Phải duyệt toàn bộ danh sách ⇐ Dùng cây nhị phân?
2. Bảng băm - Bảng băm với danh sách tuyến tính
   - Giảm thời gian duyệt thư mục
   - Đòi hỏi có một hàm băm hiệu quả

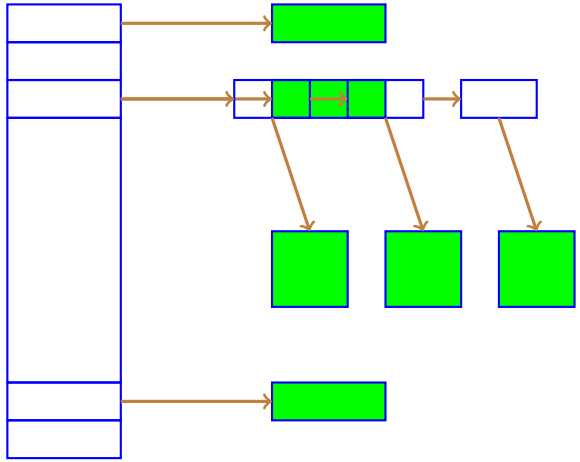
$$h(Name) = \frac{\sum_{i=1}^{Len(Name)} ASCII(Name[i])}{Table\_Size}$$

   - Vấn đề đụng độ ← hàm băm trả về cùng một kết quả với 2 tên file khác nhau
   - Vấn đề kích thước cố định → Tăng kích thước phải tính toán lại những phần đã tồn tại

Danh sách tuyến tính



Bảng băm



Các phương pháp

Mục đích

- Tăng hiệu năng truy nhập tuần tự
- Dễ dàng truy nhập ngẫu nhiên tới file
- Dễ dàng quản lý file

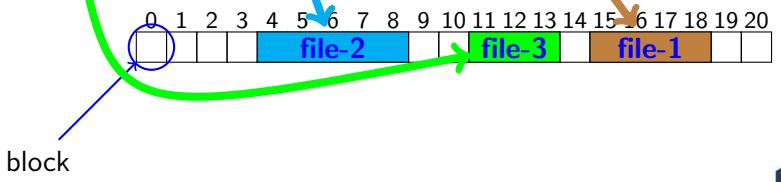
Phương pháp

1. Phân phối liên tục (*Continuous Allocation*)
2. Phân phối liên kết (*Linked List Allocation*)
3. Phân phối chỉ mục (*Indexed Allocation*)

Phân phối liên tục

Nguyên tắc: File được phân phối các khối nhớ liên tiếp nhau

File	Pos	Size
file-1	15	4
file-2	4	5
file-3	11	3
Directory		



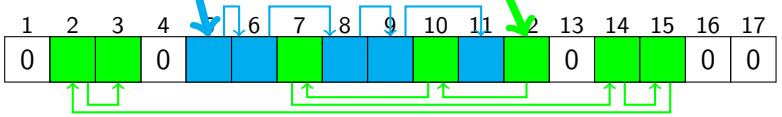
Phân phối liên tục (tiếp tục)

- File có độ dài n và bắt đầu ở khối b sẽ chiếm các khối $b, b+1, \ldots, b+n-1$
  - Hai khối b và $b+1$ liên tiếp nhau
    $\Rightarrow$ Không phải dịch chuyển đầu từ khi đọc (*trừ sector cuối*)
    $\Rightarrow$ Tốc độ truy nhập nhanh
  - Cho phép truy nhập trực tiếp khối i của file
    $\Rightarrow$ truy nhập khối $b+i-1$ trên thiết bị lưu trữ
- Lựa chọn vùng trống khi có yêu cầu lưu trữ?
  - Các chiến lược *First-Fit* /*Worst Fit* /*Best Fit*
  - Hiện tượng phân đoạn ngoài
- Khó khăn khi muốn tăng kích thước của file

Phân phối liên kết

Nguyên tắc: File được phân phối các khối nhớ không liên tục. Cuối mỗi khối là con trỏ, trỏ tới khối tiếp theo

File	Pos	End
abc	12	3
def	5	11
Directory		



File abc gồm 7 khối: 12, 10, 7, 14, 15, 2, 3
File def gồm 5 khối: 5, 6, 8, 9, 11

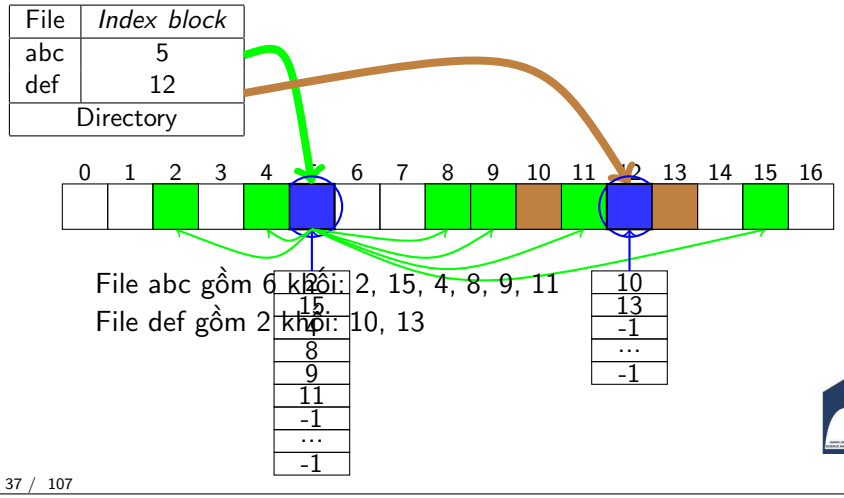
Phân phối liên kết(tiếp tục)

- Chỉ áp dụng hiệu quả cho các file truy nhập tuần tự
- Để truy nhập khối thứ n, phải duyệt qua $n-1$ khối trước đó
  - Các khối không liên tục, phải định vị lại đầu từ
  - Tốc độ truy nhập chậm
- Các khối trong file được liên kết bởi con trỏ. Nếu con trỏ lỗi?
  - Bị mất dữ liệu do mất liên kết tới khối
  - Liên kết tới khối không có dữ liệu hoặc khối của file khác

  Giải quyết: Sử dụng nhiều con trỏ trong mỗi khối $\Rightarrow$Tốn nhớ
- **Áp dụng:** FAT
  - Được sử dụng như danh sách liên kết
  - Gồm nhiều phần tử, mỗi phần tử ứng với một khối
  - Mỗi phần tử trong FAT, chứa khối tiếp theo của file
  - Khối cuối cùng có giá trị đặc biệt (*FFFF*)
  - Khối bị hỏng có giá trị (FFF7)
  - Khối chưa sử dụng có giá trị (0)
  - Trường vị trí trong bản ghi file, chứa khối đầu tiên của file

Phân phối chỉ mục

Nguyên tắc: Mỗi file có một khối chỉ mục chính (*index block*) chứa danh sách các khối dữ liệu của file

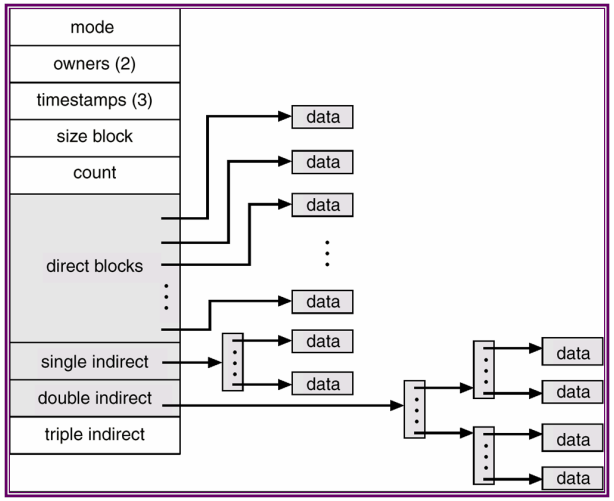


Phân phối Phân phối chỉ mục (tiếp tục)

- Phần tử thứ *i* của khối chỉ mục trỏ tới khối thứ *i* của file
  - Đọc khối *i* dùng con trỏ được khi tại p/tử *i* của khối chỉ mục
- Tạo file, các phần tử của khối chỉ mục có giá trị *null* (*-1*)
- Cần thêm khối *i*, địa chỉ khối được cấp, được đưa vào p/tử *i*
- Nhận xét
  - Không gây hiện tượng phân đoạn ngoài
  - Cho phép truy nhập trực tiếp
  - Cần khối chỉ mục: file có k/thước nhỏ, vẫn cần 2 khối
    - Khối cho dữ liệu
    - Khối chi khối chỉ mục (*chỉ dùng 1 phần tử*)

  Giải quyết: Giảm kích thước khối ⇒ Giảm phí tổn bộ nhớ
  ⇒ Vấn đề về kích thước file có thể lưu trữ.
- Sơ đồ liên kết
  - Liên kết các khối chỉ mục lại
  - P/tử cuối của khối chỉ mục trỏ tới khối chỉ mục khác nếu cần
- Index nhiều mức
  - Dùng một khối chỉ mục trỏ tới các khối chỉ mục khác

Sơ đồ kết hợp (UNIX)



- 12 *direct block* trỏ tới data block
- *Single indirect block* chứa địa chỉ khối *direct block*
- *Double indirect block* chứa địa chỉ khối *Single indirect block*
- *Triple indirect block* chứa địa chỉ khối *Double indirect*

2 Cài đặt hệ thống file

- Cài đặt thư mục
- Các phương pháp phân phối vùng lưu trữ
- Quản lý vùng lưu trữ tự do

Phương pháp

1. Bit vector
   - Mỗi block thể hiện bởi 1 bit (*1: free; 0: allocated*)
   - Dễ dàng tìm ra n khối nhớ liên tục
   - Cần có lệnh cho phép làm việc với bit
2. Danh sách liên kết (*link list*)
   - Lưu giữ con trỏ tới khối đĩa trống đầu tiên
   - Khối nhớ này chứa con trỏ trở tới khối đĩa trống tiếp theo
   - Không hiệu quả khi duyệt danh sách
3. Nhóm (*Grouping*)
   - Lưu trữ địa chỉ n khối tự do trong khối tự do đầu tiên
   - $n-1$ khối đầu tự do, khối n chứa đ/chỉ của n khối tự do tiếp
   - Ưu điểm: Tìm vùng nhớ tự do nhanh chóng
4. Bộ đếm (*Counting*)
   - Do các khối nhớ liên tục được c/cấp và g/phóng đồng thời
   - **Nguyên tắc:** Lưu địa chỉ khối nhớ tự do đầu tiên và kích thước vùng nhớ liên tục trong DS quản lý vùng trống
   - Hiệu quả khi bộ đếm lớn hơn 1

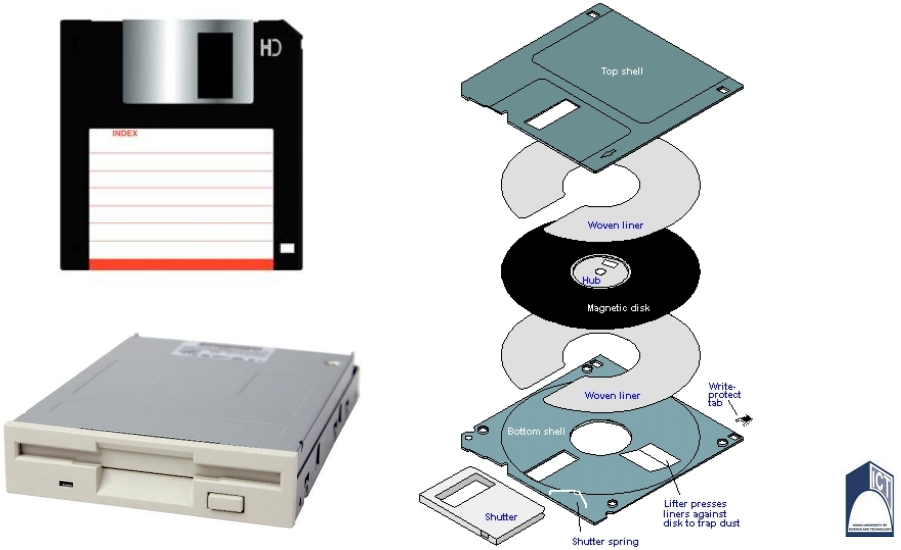
Nội dung chính

3. Tổ chức thông tin trên đĩa từ

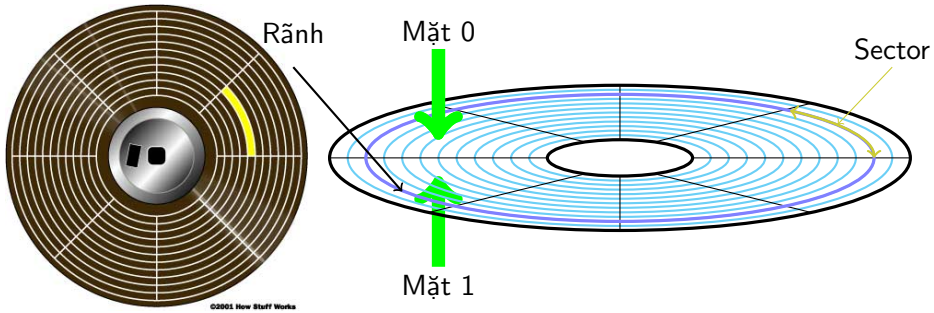
- Cấu trúc vật lý của đĩa
- Cấu trúc logic của đĩa

Đĩa mềm $5\frac{1}{4}$

Đĩa mềm $3\frac{1}{2}$



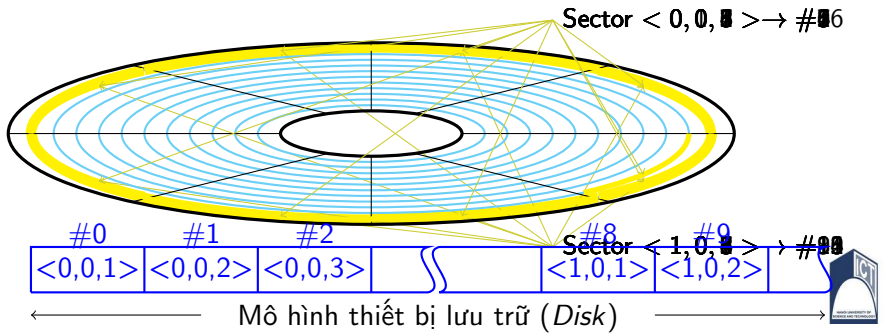
Cấu trúc vật lý đĩa mềm



- Mặt đĩa. Mỗi mặt đĩa được đọc bởi một đầu đọc (*Header*)
  - Các đầu từ được đánh số **0**, 1
- Rãnh đĩa (*Track*): Các vòng tròn đồng tâm
  - Được đánh số **0**, 1,... từ ngoài vào trong
- Cung từ (*Sector*)
  - Được đánh số **1**, 2,...

Định vị thông tin trên đĩa mềm

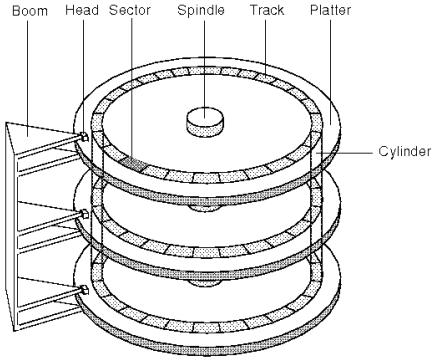
- Sector đơn vị thông tin hệ thống dùng làm việc với đĩa
- Sector xác định qua tọa độ 3 chiều: Header, Track, Sector
  - Ví dụ: Boot Sector của đĩa mềm: Sector <0, 0, 1>
- Sector được xác định qua số hiệu sector (*tọa độ 1 chiều*)
  - Vị trí tương đối so với sector đầu tiên của đĩa



Đĩa cứng

Đĩa cứng

Cấu trúc vật lý đĩa cứng



Cấu trúc

- Gồm nhiều mặt đĩa, được đánh số từ 0,1
- Các rãnh cùng bán kính tạo nên cylinder, được đánh số từ 0, 1,..
- Các sector trên mỗi mặt của mỗi cylinder, được đánh số từ 1,2,...

Định vị thông tin

- Tọa độ 3 chiều (H, C, S)
- Tọa độ 1 chiều: Số hiệu sector
  - Nguyên tắc như với đĩa mềm: Sector→Header→Cylinder

Truy nhập sector trên đĩa

- Sector là đơn vị thông tin máy tính dùng để làm việc với đĩa từ
- Có thể truy nhập (*đọc/ghi/format/...*) tới từng sector
- Truy nhập sử dụng ngắt BIOS 13h (*chức năng 2, 3, 5,...*)
  - Không phụ thuộc hệ điều hành
  - Sector được xác định theo địa chỉ <H,C,S>
- Truy nhập sử dụng lời gọi hệ thống
  - Ngắt của hệ điều hành
    - **Ví dụ:** MSDOS cung cấp ngắt 25h/26h cho phép đọc/ghi các sector theo địa chỉ tuyến tính
  - Sử dụng hàm WIN32 API
    - CreateFile()/ReadFile()/WriteFile()...

Sử dụng ngắt 13h

Thanh ghi	Ý nghĩa
AH	**2h:Đọc secror; 3h: Ghi Sector**
AL	Số sector cần đọc Các sector phải trên cùng một mặt, một rãnh
DH	Số hiệu mặt đĩa
DL	Số hiệu ổ đĩa. 0h:A; 80h: Đĩa cứng thứ nhất; 81h Đĩa cứng thứ 2
CH	Số hiệu Track/Cylinder (*Sử dụng 10 bit, trong đó lấy 2 bit cao của CL*)
CL	Số hiệu sector (*chỉ sử dụng 6 bit thấp*)
ES:BX	Trỏ tới vùng đệm, nơi sẽ chứa dữ liệu đọc đươc (*khi AH=2h*) hoặc dữ liệu ghi ra đĩa (*Khi AH=3h*)
CarryFlag	CF=0 không có lỗi; CL chứa số sector đọc được CF=1 Có lỗi, AH chứa mã lỗi

WinXP hạn chế sử dụng ngắt 13h để truy nhập trực tiếp

Sử dụng ngắt 13h (Ví dụ)

```
#include <stdio.h>
#include <dos.h>
int main(int argc, char *argv[]){
  union REGS regs;
  struct SREGS sregs;
  int Buf[512];
  int i;
  regs.h.ah = 0x02;  regs.h.al = 0x01;
  regs.h.dh = 0x00;  regs.h.dl = 0x80;
  regs.h.ch = 0x00;  regs.h.cl = 0x01;
  regs.x.bx = FP_OFF(Buf);
  sregs.es  = FP_SEG(Buf);
  int86x(0x13,&regs,&regs,&sregs);
  for(i=0;i<512;i++)  printf("%4X",Buf[i]);
  return 0;
}
```


Sử dụng WIN32 API

- **HANDLE CreateFile(...)**: Mở file/thiết bị vào ra
  - **LPCTSTR lpFileName**, ⇒ Tên file/thiết bị vào ra
    - "\\\\.*C* :" Phân vùng / Ổ đĩa C
    - "\\\\.*PhysicalDrive*0" Ổ đĩa cứng thứ nhất
  - **DWORD** dwDesiredAccess,⇒ Thao tác với thiết bị
  - **DWORD** dwShareMode,⇒ Cho phép dùng chung
  - **LPSECURITY_ATTRIBUTES** lpSecurityAttributes (*NULL*),
  - **DWORD** dwCreationDisposition,⇒ Hành động thực hiện
  - **DWORD** dwFlagsAndAttributes, ⇒ Thuộc tính
  - **HANDLE** hTemplateFile *(NULL)*
- **BOOL ReadFile(...)**
  - **HANDLE** hFile,⇒File muốn đọc
  - **LPVOID** lpBuffer, ⇒ Vùng đệm chứa dữ liệu
  - **DWORD** nNumberOfBytesToRead,⇒, số byte cần đọc
  - **LPDWORD** lpNumberOfBytesRead,⇒ số byte đọc được
  - **LPOVERLAPPED** lpOverlapped (*NULL*)
- **BOOL WriteFile(...)** ⇒Tham số tương tự ReadFile()

Sử dụng WIN32 API (Ví dụ)

```
#include <windows.h>
#include <stdio.h>
int main(int argc, char *argv[]){
  HANDLE hDisk;
  BYTE Buf[512];
  int byteread,i;

  hDisk=CreateFile("\\\\.\\PhysicalDrive0",GENERIC_READ,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL, OPEN_EXISTING,0,NULL);
  if (hDisk==INVALID_HANDLE_VALUE) printf("Loi thiet bi");
  else {
     ReadFile(hDisk,Buf,512,&byteread,NULL);
     for(i=0;i<512;i++)  printf("%4X",Buf[i]);
     CloseHandle(hDisk);
  }
  return 0;
}
```


Kết quả thực hiện

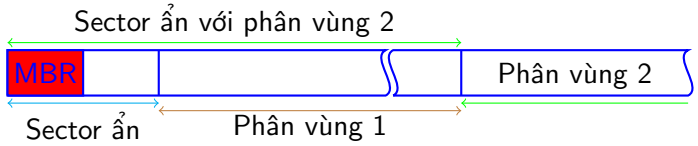
```
33 C0 8E D0 BC 00 7C FB 50 07 50 1F FC BE 1B 7C BF 1B 06 50
57 B9 E5 01 F3 A4 CB BD BE 07 B1 04 38 6E 00 7C 09 75 13 83
C5 10 E2 F4 CD 18 8B F5 83 C6 10 49 74 19 38 2C 74 F6 A0 B5
07 B4 07 8B F0 AC 3C 00 74 FC BB 07 00 B4 0E CD 10 EB F2 88
4E 10 E8 46 00 73 2A FE 46 10 80 7E 04 0B 74 0B 80 7E 04 0C
74 05 A0 B6 07 75 D2 80 46 02 06 83 46 08 06 83 56 0A 00 E8
21 00 73 05 A0 B6 07 EB BC 81 3E FE 7D 55 AA 74 0B 80 7E 10
00 74 C8 A0 B7 07 EB A9 8B FC 1E 57 8B F5 CB BF 05 00 8A 56
00 B4 08 CD 13 72 23 8A C1 24 3F 98 8A DE 8A FC 43 F7 E3 8B
D1 86 D6 B1 06 D2 EE 42 F7 E2 39 56 0A 77 23 72 05 39 46 08
73 1C B8 01 02 BB 00 7C 8B 4E 02 8B 56 00 CD 13 73 51 4F 74
4E 32 E4 8A 56 00 CD 13 EB E4 8A 56 00 60 BB AA 55 B4 41 CD
13 72 36 81 FB 55 AA 75 30 F6 C1 01 74 2B 61 60 6A 00 6A 00
FF 76 0A FF 76 08 6A 00 68 00 7C 6A 01 6A 10 B4 42 8B F4 CD
13 61 61 73 0E 4F 74 0B 32 E4 8A 56 00 CD 13 EB D6 61 F9 C3
49 6E 76 61 6C 69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61
62 6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67 20 6F 70
65 72 61 74 69 6E 67 20 73 79 73 74 65 6D 00 4D 69 73 73 69
6E 67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65 6D 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2C 44 63
0A 08 0B 08 00 00 80 01 01 00 07 FE FF FF 3F 00 00 00 2C 92
00 02 00 00 C1 FF 0F FE FF FF 31 41 8A 03 0E D3 1D 01 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 55 AA
```

3 Tổ chức thông tin trên đĩa từ

- Cấu trúc vật lý của đĩa
- Cấu trúc logic của đĩa

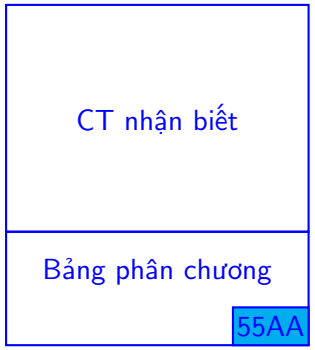
Cấu trúc logic

- Đĩa mềm: Mỗi hệ điều hành có một chiến lược quản lý riêng
- Đĩa cứng (*Có dung lượng lớn*)
  - Được chia thành nhiều phân vùng (Partitions, Volumes,..)
    - Mỗi vùng là tập hợp các *Cylinder* liên tiếp nhau
    - Người dùng ấn định kích thước (*Ví dụ dùng: fdisk*)
  - Mỗi phân vùng có thể được quản lý bởi một HĐH riêng
    - HĐH *format* phân vùng theo định dạng được sử dụng
    - Tồn tại nhiều hệ thống khác nhau: FAT, NTFS, EXT3,...
  - Trước tất cả các phân vùng là các sector bị che
    - *Master Boot Record* (**MBR**): Sector đầu tiên của đĩa



Master Boor Record

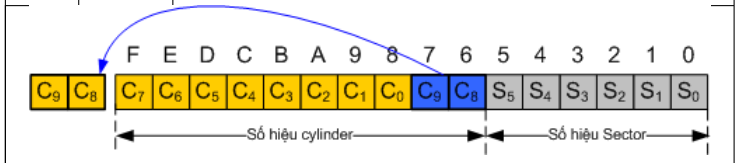
- Sector quan trọng nhất của đĩa
- Là sector đầu tiên trên đĩa (*Số hiệu 0 hoặc địa chỉ <0, 0, 1>*)
- Cấu trúc gồm 3 phần



- **Chương trình nhận biết**
  - Đọc bảng phân chương để biết
    - Vị trí các phân vùng
    - Phân vùng tích cực (*chứa HĐH*)
  - Đọc và thực hiện sector đầu tiên của phân vùng tích cực
- **Bảng phân chương** (*64bytes*)
  - Gồm 4 phần tử, mỗi phần tử 16 bytes
  - Mỗi phần tử chứa thông tin một vùng
    - Vị trí, kích thước, hệ thống chiếm giữ
- **Chữ ký hệ thống** (*luôn là 55AA*)

Cấu trúc một phần tử bảng phân chương

	Stt	Ofs	Size	Ý nghĩa
	1	0	1B	Phân vùng tích cực? 80h nếu đúng; 0: Data
địa chỉ đầu	2	1	1B	Số hiệu mặt đĩa đầu của phân vùng
	3	2	1W	Số hiệu sector và cylinder đầu của phân vùng
	4	4	1B	Mã nhận diện hệ thống. 05/0F: Partition mở rộng; 06:Big Dos; 07:NTFS; 0B: FAT32,..
đ/c cuối	5	5	1B	Số hiệu đầu đọc cuối
	6	6	1W	Số hiệu sector và cylinder cuối của phân vùng. *(Số hiệu sector chỉ dùng 6 bit thấp)*
	7	8	1DW	Địa chỉ đầu, tính theo số hiệu sector
	8	12	1DW	Số sector trong phân vùng



Ví dụ 1

00 01 01 00	07 FE 3F F8	3F 00 00 00	7A 09 3D 00
80 00 01 F9	0B FE BF 30	B9 09 3D 00	38 7B 4C 00
00 00 81 EB	0F FE FF FF	2B 1D B7 00	72 13 7A 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
55 AA			

Giải mã

Boot	Vị trí đầu			Vị trí cuối			#sector	số sector
	Hdr	Cyl	Sec	HdR	Cyl	Sec		
No	1	0	1	254	248	63	63	4000122
Yes	0	249	1	254	560	63	4000185	5012280
No	0	747	1	254	1023	63	12000555	8000370
-	0	0	0	0	0	0	0	0

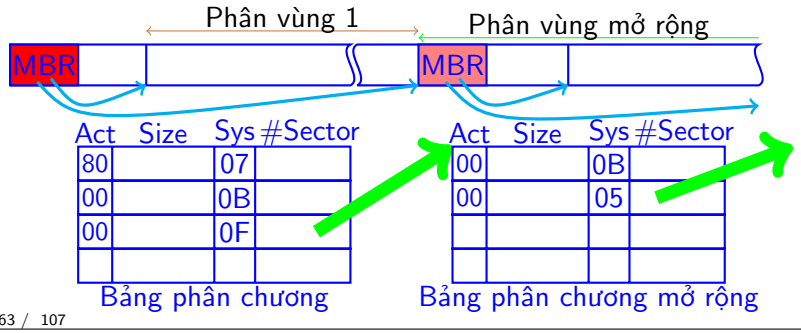
Ví dụ 2

```
80  01  01  00  07  FE  FF  FF  3F  00  00  00  2C  92  00  02
00  00  C1  FF  0F  FE  FF  FF  31  41  8A  03  0E  D3  1D  01
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
55  AA
```

Active	Begin Hdr	Begin Cyl	Begin Sct	Sys	End Hdr	End Cyl	End Sct	Relative Sector	Number Of Sector
YES	1	0(0)	1	07	254	1023(2090)	63	63	33591852
NO	0	1023(3697)	1	0F	254	1023(4862)	63	59392305	18731790
NO	0	0(0)	0	00	0	0	0	0	0
NO	0	0(0)	0	00	0	0	0	0	0

Bảng phân chương mở rộng

- Khi trường nhận diện có giá trị 05 hoặc 0F, partition tương ứng là partition mở rộng
- Partition mở rộng được tổ chức như một đĩa cứng vật lý
  - Sector đầu tiên là MBR, chứa thông tin về các phân vùng trong partition mở rộng này
    - Các phần tử trong partition mở rộng có thể là partition rộng
  - Cho phép tạo hơn 4 ổ đĩa logic



Ví dụ về bảng phân chương mở rộng 1

```
80  01  01  00  07  EF  FF  FF  3F  00  00  00  11  2F  F7  01
00  00  C1  FF  0F  EF  FF  FF  50  2F  F7  01  B0  23  B1  02
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
55  AA
```

Active	Begin Hdr	Begin Cyl	Begin Sct	Sys	End Hdr	End Cyl	End Sct	Relative Sector	Number Of Sector
YES	1	0	1	07	239	1023	63	63	32976657
NO	0	1023	1	0F	239	1023	63	32976720	45163440
NO	0	0	0	00	0	0	0	0	0
NO	0	0	0	00	0	0	0	0	0

Ví dụ về bảng phân chương mở rộng 2

```
Extended Partition (Sector number 32976720)...
 00   01   C1   FF   06   EF   FF   FF   3F   00   00   00   51   E8   76   01
 00   00   C1   FF   05   EF   FF   FF   90   E8   76   01   20   3B   3A   01
 00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
 00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
 55   AA
```

Active	Begin			Sys	End			Relative	Number
	Hdr	Cyl	Sct		Hdr	Cyl	Sct	Sector	Of Sector
NO	1	1023	1	06	239	1023	63	63	24569937
NO	0	1023	1	05	239	1023	63	24570000	20593440
NO	0	0	0	00	0	0	0	0	0
NO	0	0	0	00	0	0	0	0	0

Ví dụ về bảng phân chương mở rộng 3

```
Extended Partition (Sector number 57546720)...
 00   01   C1   FF   0B   EF   FF   FF   3F   00   00   00   E1   3A   3A   01
 00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
 00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
 00   00   00   00   00   00   00   00   00   00   00   00   00   00   00   00
 55   AA
```

Active	Begin			Sys	End			Relative	Number
	Hdr	Cyl	Sct		Hdr	Cyl	Sct	Sector	Of Sector
NO	1	1023	1	0B	239	1023	63	63	20593377
NO	0	0	0	00	0	0	0	0	0
NO	0	0	0	00	0	0	0	0	0
NO	0	0	0	00	0	0	0	0	0

Nội dung chính

Các hệ thống file

Tồn tại nhiều hệ thống file khác nhau

- Hệ thống FAT
  - FAT 12/ FAT16 dùng cho MSDOS
  - FAT32 dùng từ WIN98
  - 12/16/32: Số bit dùng để định danh cluster
- Hệ thống NTFS
  - Sử dụng trong WINNT, WIN2000
  - Dùng 64 bit để xác định một cluster
  - Ưu việt hơn FAT trong bảo mật, mã hóa, nén dữ liệu,...
- Hệ thống EXT3
  - Sử dụng trong Linux
- Hệ thống CDFS
  - Hệ thống quản lý file trong CDROM
  - Hạn chế về độ sâu cây thư mục và kích thước tên
- Hệ thốngs UDF
  - Phát triển từ CDFS cho DVD-ROM, hỗ trợ tên file dài

Cấu trúc phân vùng cho FAT

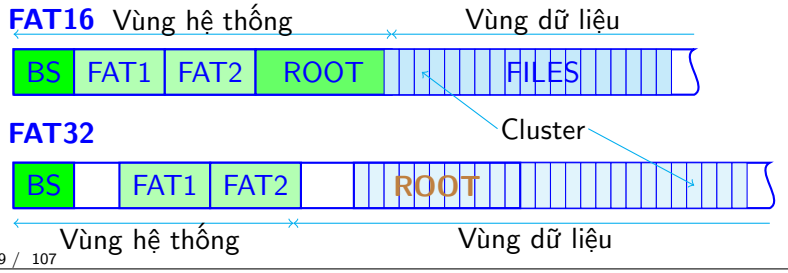
FAT12/16

- Số cluster lớn nhất FAT12: $2^{12} - 18$; *FAT*16 : $2^{16} - 18$
- K/thước max: FAT12: 32MB; FAT16: 2GB/4GB (32K/64K Cluster)

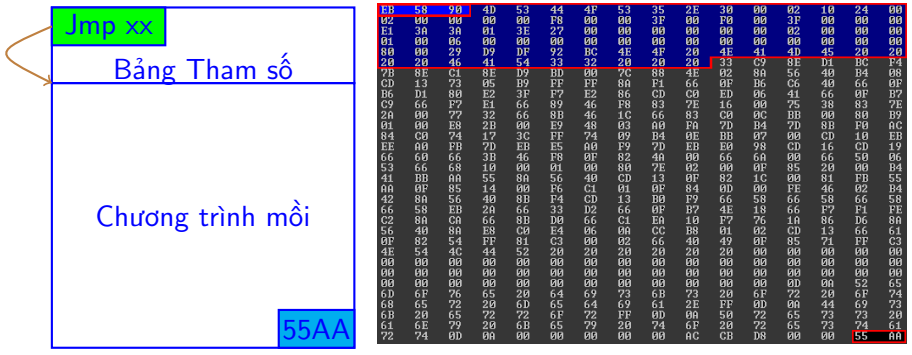
FAT32

- Chỉ dùng 28 bit ⇒ Số cluster lớn nhất $2^{28} - 18$
- K/thước max: 2TB/8GB/16TB (8KB/32KB/64KB Cluster)

Cấu trúc logic của hệ thống FAT



Cấu trúc



- Sector đầu tiên của phân vùng
- Cấu trúc gồm 3 phần
  - Bảng tham số đĩa (*BPB: Bios Parameter Block*)
  - Chương trình mồi (*Boot strap loader*)
  - Chữ ký hệ thống (*luôn là 55AA*)

Cấu trúc bảng tham số đĩa - Phần chung

Stt	Ofs	Kt	Giá trị mẫu	Ý nghĩa
1	0	3B	EB 3C 90	Nhảy đến đầu chương trình mồi
2	3	8B	MSDOS5.0	Tên hệ thống file đã format đĩa
3	11	1W	00 02	K/thước 1 sector, thường là 512
4	13	1B	40	Số sector cho một cluster (*32K-Cluster*)
5	14	1W	01 00	Số scts đứng trước FAT/Số scts để dành
6	16	1B	02	Số bảng FAT
7	17	1W	00 02	Số phần tử của ROOT. FAT32: 00 00
8	19	1W	00 00	$\sum$ sector trên đĩa (< 32M) hoặc 0000
9	21	1B	F8	Khuôn dạng đĩa (F8:HD, F0: Đĩa1.44M)
10	22	1W	D1 09	Số sector cho một bảng FAT(*209*)
11	24	1W	3F 00	Số sector cho một rãnh (*63*)
12	26	1W	40 00	Số đầu đọc ghi (*64*)
13	28	1DW	3F 00 00 00	Số sector ẩn- Sectors trước volume (*63*)
14	32	1DW	41 0C 34 00	Tổng số sector trên đĩa (*3411009*)

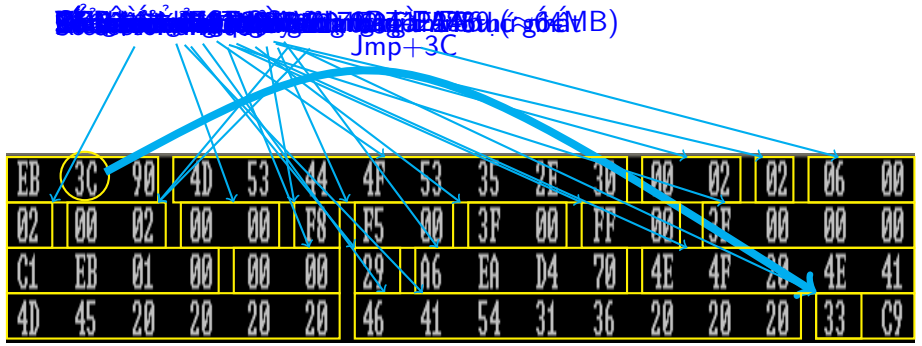
Cấu trúc bảng tham số đĩa - Phần dành cho FAT12/FAT16

Stt	Ofs	Kt	Giá trị mẫu	Ý nghĩa
15	36	1B	80h	Số hiệu ổ đĩa vật lý 0: ổ A; 80h: ổ C
16	37	1B	00	Để dành/Byte cao cho trường *#ổ đĩa*
17	38	1B	29h	Boot sector mở rộng 29h
18	39	1DW	D513 5B24	Volumn Serial number(245B-13D5)
19	43	11B	NO NAME	Volumn Label: nhãn đĩa (*không dùng*)
20	54	8B	FAT16	Để dành, thường là đoạn text miêu tả dạng FAT
21	62	-		Bootstrap loader

Ví dụ

```
EB 3C 90 4D 53 44 4F 53 35 2E 30 00 02 02 06 00
02 00 02 00 00 F8 F5 00 3F 00 FF 00 3F 00 00 00
C1 EB 01 00 00 00 29 A6 EA D4 70 4E 4F 20 4E 41
4D 45 20 20 20 20 46 41 54 31 36 20 20 20 33 C9
```


Ví dụ giả mã bảng tham số đĩa của FAT16



Cấu trúc bảng tham số đĩa - Phần dành cho FAT32

Stt	Ofs	Kt	Giá trị mẫu	Ý nghĩa
15	36	1DW	C9 03 00 00	Tổng số sector cho bảng FAT
16	40	1W	00 00	Flags: #FAT chính(*Không dùng*)
17	42	1W	00 00	Version: Phiên bản FAT32 (*Không dùng*)
18	44	1DW	02 00 00 00	Số hiệu cluster bắt đầu của ROOT
19	48	1W	01 00	#sector chứa File System information
20	50	1W	06 00	Số hiệu sector dùng backup Bootsector
21	52	12B	00 . . . 00	Để dành
22	64	1B	00	Số hiệu ổ đĩa vật lý 0: ổ A; 80h: ổ C
23	65	1B	00	Để dành/Byte cao cho trường #Driver
24	66	1B	29	Boot sector mở rộng. Luôn có giá trị 29h
25	67	1DW	62 0E 18 66	Volumn Serial number
26	71	11B	NO NAME	Volumn Label: Nhãn đĩa (*Ko s/dụng*)
27	82	8B	FAT32	Để dành, thường là đoạn text miêu tả dạng FAT

Ví dụ Boot sector của một hệ thống dùng FAT32

```
EB 58 90 4D 53 44 4F 53 35 2E 30 00 02 10 24 00
02 00 00 00 00 F8 00 00 3F 00 00 00 F0 00 3F 00 00 00 00 00
E1 3A 3A 01 3E 27 00 00 00 00 00 00 00 02 00 00 00 00
01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 29 D9 DF 92 BC 4E 4F 20 4E 41 4D 45 20 20
20 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4
7B 8E C1 8E D9 BD 00 7C 88 4E 02 8A 56 40 B4 08
CD 13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F
B6 D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7
C9 66 F7 E1 66 89 46 F8 83 7E 16 00 75 38 83 7E
2A 00 77 32 66 8B 46 1C 66 83 C0 0C BB 00 80 B9
01 00 E8 2B 00 E9 48 03 A0 FA 7D B4 7D 8B F0 AC
84 C0 74 17 3C FF 74 09 B4 0E BB 07 00 CD 10 EB
EE A0 FB 7D EB E5 A0 F9 7D EB E0 98 CD 16 CD 19
66 60 66 3B 46 F8 0F 82 4A 00 66 6A 00 66 50 06
53 66 68 10 00 01 00 80 7E 02 00 0F 85 20 00 B4
41 BB AA 55 8A 56 40 CD 13 0F 82 1C 00 81 FB 55
AA 0F 85 14 00 F6 C1 01 0F 84 0D 00 FE 46 02 B4
42 8A 56 40 8B F4 CD 13 B0 F9 66 58 66 58 66 58
66 58 EB 2A 66 33 D2 66 0F B7 4E 18 66 F7 F1 FE
C2 8A CA 66 8B D0 66 C1 EA 10 F7 76 1A 86 D6 8A
56 40 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 66 61
0F 82 54 FF 81 C3 00 02 66 40 49 0F 85 71 FF C3
4E 54 4C 44 52 20 20 20 20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 0D 0A 52 65
6D 6F 76 65 20 64 69 73 6B 73 20 6F 72 20 6F 74
68 65 72 20 6D 65 64 69 61 2E FF 0D 0A 44 69 73
6B 20 65 72 72 6F 72 FF 0D 0A 50 72 65 73 73 20
61 6E 79 20 6B 65 79 20 74 6F 20 72 65 73 74 61
72 74 0D 0A 00 00 00 00 00 AC CB D8 00 00 55 AA
```

Kết quả giải mã hệ thống FAT32 bằng chương trình

```
BIOS PARAMETER BLOCK (BPB)...
OEM Name                              :   MSDOS5.0
Bytes per sector                      :           512
Sectors per cluster                   :            16
Sectorss before the first FAT         :            36
Number of copies of FAT               :             2
Media Desctiptor                      :           F8h
Sectors per Tracks                    :            63
Number of Header                      :           240
Number of Hiden Scts in Volume        :            63
Number of Sectors in Volume           :      20593377
Number of Sectors per FAT             :         10046
Cluster num. of start of ROOT         :             2
Sct number of FileSystem Info         :             1
Sct number of Boot backup sct         :             6
Logical drive number of Volume        :           80h
Extend BPB Signature                  :           29h
Serial Number of Volume               :     BC92-DFD9
Volumn lable                          :      NO NAME
FAT Type                              :         FAT32
Boot signature                        :         55 AA
```


File System Information Sector

- Thường là Sector thứ 2 của Volume
  - Ngay sau Boot sector *(Sector số hiệu 1)*
- Cấu trúc

Stt	Ofs	Size	Ý nghĩa
1	0	1DW	Chữ ký thứ nhất của FSInfo sector. Giá trị các byte theo thứ tự: 52h 52h 61h 41h
2	4	480B	Không rõ, thường chứa giá trị 00
3	484	1DW	Chữ ký của File System Information Sector. Giá trị các byte theo thứ tự: 72h 72h 41h 61h
4	488	1DW	Số cluster tự do. *-1* nếu không xác định
5	492	1DW	Số hiệu của cluster vừa mới được cung cấp
6	496	12B	Để dành
7	508	2B	Không xác định, thường bằng 0
8	510	2B	Chữ ký Bootsector. Có giá trị 55 AA

File system information sector của một volume dùng FAT32

```
52 52 61 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FILE SYSTEM INFO:...
First signature                       : 41615252h
File System Info Signature            : 61417272h
Number of Free Clusters               :      786100
#Cluster recently Allocation          :        2469
Boot signature                        :       55 AA

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 72 72 41 61 B4 FE 0B 00 A5 09 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 55 AA
```


Cấu trúc bảng tham số đĩa cho hệ thống NTFS 1

Stt	Ofs	Kt	Giá trị mẫu	Ý nghĩa
1	0	3B	EB 52 90	Nhảy đến đầu chương trình mồi
2	3	8B	NTFS	Tên hệ thống file đã format đĩa
3	11	1W	00 02	Bytes per Sector
4	13	1B	08	Sectors per Cluster (*4K-Cluster*)
5	14	1W	00 00	Reserved sectors. Allways zero
6	16	1B	00	Allways 0 (**FAT**: Số bảng FAT)
7	17	1W	00 00	Allways 0 (**FAT**: Số p/tử của ROOT)
8	19	1W	00 00	Not used by NTFS (**FAT**:K/thước đĩa)
9	21	1B	F8	Media Type
10	22	1W	00 00	Allway 0 (**FAT**:Sectors cho FAT)
11	24	1W	3F 00	Sector per Track (*63*)
12	26	1W	FF 00	Number of Head (*255*)
13	28	1DW	3F 00 00 00	Hidden sectors (*63*)
14	32	1DW	00 00 00 00	Not used by NTFS (**FAT**: $\sum$sectors)

Cấu trúc bảng tham số đĩa cho hệ thống NTFS 2

Stt	Ofs	Kt	Giá trị mẫu	Ý nghĩa
15	36	1DW	80 00 80 00	Not used by NTFS(**FAT**:Tổng số sectors cho FAT)
16	40	1LCN	2B 92 00 02 00 00 00 00	Total sectors (*LCN:LONGLONG*) (*33591851*)
17	48	1LCN	00 00 0C 00 00 00 00 00	Logical cluster number for MFT (*786432*)
18	56	1LCN	22 09 20 00 00 00 00 00	Logical #cluster for MFT mirroring (*2099490*)
19	64	1DW	F6 00 00 00	Clusters per file record segment (246)
20	68	1DW	01 00 00 00	Clusters per index block (1)
21	72	1LCN	A6 CA D7 C6 00 D8 6C 24	Volume serial number 246C-D800-C6D7-CAA6
22	80	1DW	00 00 00 00	Checksum
23	84	-		Bootstrap loader

Boot sectors của một đĩa dùng NTFS

```
EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00
00 00 00 00 00 F8 00 00 3F 00 FF 00 3F 00 00 00
00 00 00 00 80 00 80 00 2B 92 00 02 00 00 00 00
00 00 0C 00 00 00 00 00 22 09 20 00 00 00 00 00
F6 00 00 00 01 00 00 00 A6 CA D7 6C 00 D8 6C 24
00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB B8 C0 07
8E D8 E8 16 00 B8 00 0D 8E C0 33 DB C6 06 0E 00
10 E8 53 00 68 00 0D 68 6A 02 CB 8A 16 24 00 B4
08 CD 13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66
0F B6 D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F
B7 C9 66 F7 E1 66 A3 20 00 C3 B4 41 BB AA 55 8A
16 24 00 CD 13 72 0F 81 FB 55 AA 75 09 F6 C1 01
74 04 FE 06 14 00 C3 66 60 1E 06 66 A1 10 00 66
03 06 1C 00 66 3B 06 20 00 0F 82 3A 00 1E 66 6A
00 66 50 06 53 66 68 10 00 01 00 80 3E 14 00 00
0F 85 0C 00 E8 B3 FF 80 3E 14 00 00 0F 84 61 00
B4 42 8A 16 24 00 16 1F 8B F4 CD 13 66 58 5B 07
66 58 66 58 1F EB 2D 66 33 D2 66 0F B7 0E 18 00
66 F7 F1 FE C2 8A CA 66 8B D0 66 C1 EA 10 F7 36
1A 00 86 D6 8A 16 24 00 8A E8 C0 E4 06 0A CC B8
01 02 CD 13 0F 82 19 00 8C C0 05 20 00 8E C0 66
FF 06 10 00 FF 0E 0E 00 0F 85 6F FF 07 1F 66 61
C3 A0 F8 01 E8 09 00 A0 FB 01 E8 03 00 FB EB FE
B4 01 8B F0 AC 3C 00 74 09 B4 0E BB 07 00 CD 10
EB F2 C3 0D 0A 41 20 64 69 73 6B 20 72 65 61 64
20 65 72 72 6F 72 20 6F 63 63 75 72 72 65 64 00
0D 0A 4E 54 4C 44 52 20 69 73 20 6D 69 73 73 69
6E 67 00 0D 0A 4E 54 4C 44 52 20 69 73 20 63 6F
6D 70 72 65 73 73 65 64 00 0D 0A 50 72 65 73 73
20 43 74 72 6C 2B 41 6C 74 2B 44 65 6C 20 74 6F
20 72 65 73 74 61 72 74 0D 0A 00 00 00 00 00 00
00 00 00 00 00 00 00 00 83 A0 B3 C9 00 00 55 AA
```


Giải mã bảng tham số của đĩa dùng NTFS

```
EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00
00 00 00 00 00 F8 00 00 3F 00 FF 00 3F 00 00 00
00 00 00 00 80 00 80 00 2B 92 00 02 00 00 00 00
00 00 0C 00 00 00 00 00 22 09 20 00 00 00 00 00
F6 00 00 00 01 00 00 00 A6 CA D7 6C 00 D8 6C 24
00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB B8 C0 07
```

```
BIOS PARAMETER BLOCK (BPB)...
OEM Name                        :  NTFS
Bytes per sector                :       512
Sectors per cluster             :         8
Media Desctiptor                :       F8h
Sectors per Tracks              :        63
Number of Header                :       255
Number of Hiden Scts in Volume  :        63
Number of Sectors in Volume     :  33591851
Cluster number for MTF          :    786432
Cluster number for MTF Mirror   :   2099490
Cluster per file Record Seg.    :       246
Cluster per index block         :         1
Volume serial number            : 246C-D800-6CD7-CAA6
Checksum                        :         0
Boot signature                  :     55 AA
```


Mục đích

FAT được sử dụng để quản lý các khối nhớ (*blocks/clusters*) trong vùng dữ liệu của bộ nhớ lưu trữ

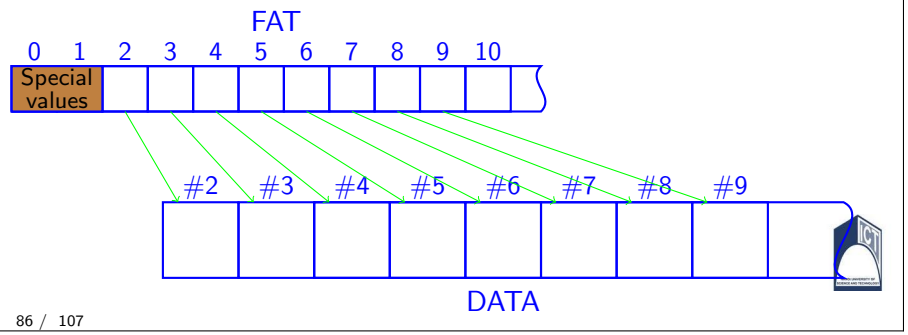
- Khối nhớ đang sử dụng
  - Phân phối cho từng file/thư mục
- Khối nhớ tự do
- Khối nhớ bị hỏng

Thực hiện như thế nào ?

Phương pháp

FAT gồm nhiều phần tử

- Mỗi phần tử có thể 12bit, 16bit, 32bit
- Mỗi phần tử ứng với 1 khối (*cluster*) trên vùng dữ liệu
  - 2 phần tử đầu (0,1) có ý nghĩa đặc biệt
    - Khuôn dạng đĩa, Bit shutdown, Bit diskerror
  - Phần tử thứ 2 ứng với cluster đầu của phần Data

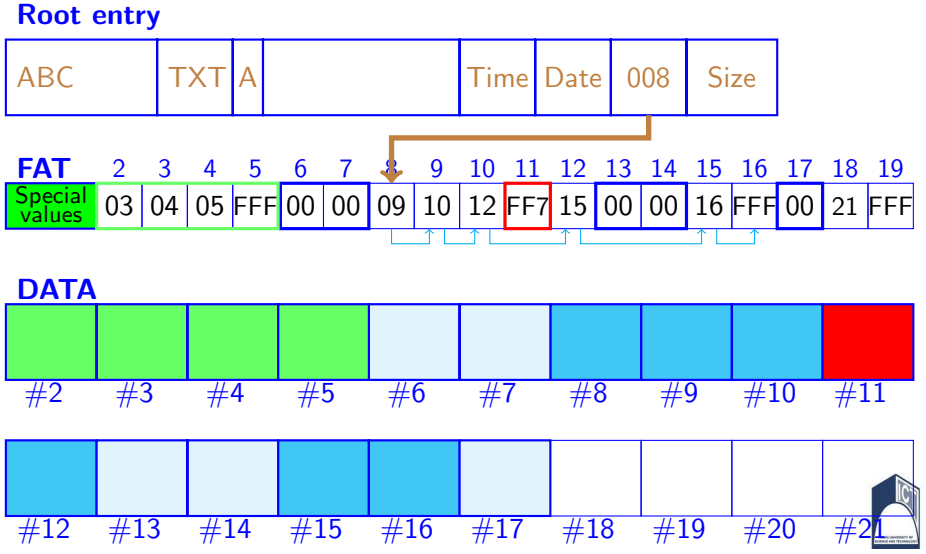


Cài đặt

Mỗi phần tử của bảng FAT mang một giá trị đặc trưng cho tính chất của cluster tương ứng

FAT[(32)16]12	Ý nghĩa
[(0000)0]000h	Cluster tương ứng tự do
[(0000)0]001h	Giá trị không sử dụng
[(0000)0]002h →[(0FFF)F]FEFh	Cluster đang được sử dụng. Giá trị đóng vai trò con trỏ, trỏ tới cluster tiếp theo của file
[(0FFF)F]FF0h →[(0FFF)F]FF6h	Các giá trị để dành, chưa được sử dụng
[(0FFF)F]FF7h	Đánh dấu cluster tương ứng bị hỏng
[(0FFF)F]FF8h→ →[(0FFF)F]FFFh	Cluster đang đc sử dụng và là cluster cuối cùng của file (*EOC:End Of Cluster chain*). Thực tế thường dùng giá trị [(0FFF)F]FFFh

Liên kết các cluster



Ví dụ: Đọc một sector của FAT32

```
#include <windows.h>
#include <stdio.h>
int main(int argc, char *argv[]){
  HANDLE hDisk;
  BYTE Buf[512];
  DWORD FAT[128];
  WORD FATAddr;   DWORD byteread, i;
  hDisk = CreateFile("\\\\.\\F:", GENERIC_READ,
             FILE_SHARE_READ | FILE_SHARE_WRITE,
             NULL, OPEN_EXISTING,0,NULL);
  ReadFile(hDisk,Buf,512,&byteread,NULL);
  memcpy(&FATAddr,&Buf[14],2);//Offset 14 Sector truoc FAT
  SetFilePointer(hDisk,FATAddr * 512, NULL,FILE_BEGIN);
  ReadFile(hDisk,FAT,512,&byteread,NULL);
  for(i=0;i<128;i++)  printf(" %08X ",FAT[i]);
  CloseHandle(hDisk);
  return 0;
}
```


Ví dụ: Sector đầu của một FAT32

A Root entry

52	45	41	44	4D	42	52	20	43	20	20	20	00	5E	B9	5A
A8	3E	A8	3E	00	00	CE	79	A4	3E	03	00	BD	0A	00	00

FAT

0FFFFFF8	FFFFFFFF	0FFFFFFF	00000004	00000005	00000006	00000007	00000008
0FFFFFFF	0000000A	0000000B	0000000C	0000000D	0FFFFFFF	0FFFFFFF	00000000
00000000	00000000	00000000	00000000	00000015	00000016	00000017	00000018
00000019	0000001A	0000001B	0000001C	0000001D	0000001E	0000001F	00000020
0FFFFFFF	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Cấu trúc thư mục gốc

- Bảng gồm các *bản ghi file*
  - Mỗi bản ghi có kích thước 32 bytes
    - Chứa các thông tin liên quan tới một file/thư mục/ nhãn đĩa
- Hệ thống FAT12/FAT16
  - Thư mục gốc nằm ngay sau các bảng FAT
  - Kích thước = Số phần tử tối đa trong thư mục gốc * $\frac{32}{512}$
- Hệ thống FAT32
  - Vị trí được xác định dựa vào BPB
    - Trường 18: Số hiệu cluster đầu của ROOT
  - Kích thước không xác định
  - Hỗ trợ tên file dài (*LFN: Long File Name*)
    - Một file có thể sử dụng nhiều hơn một phần tử

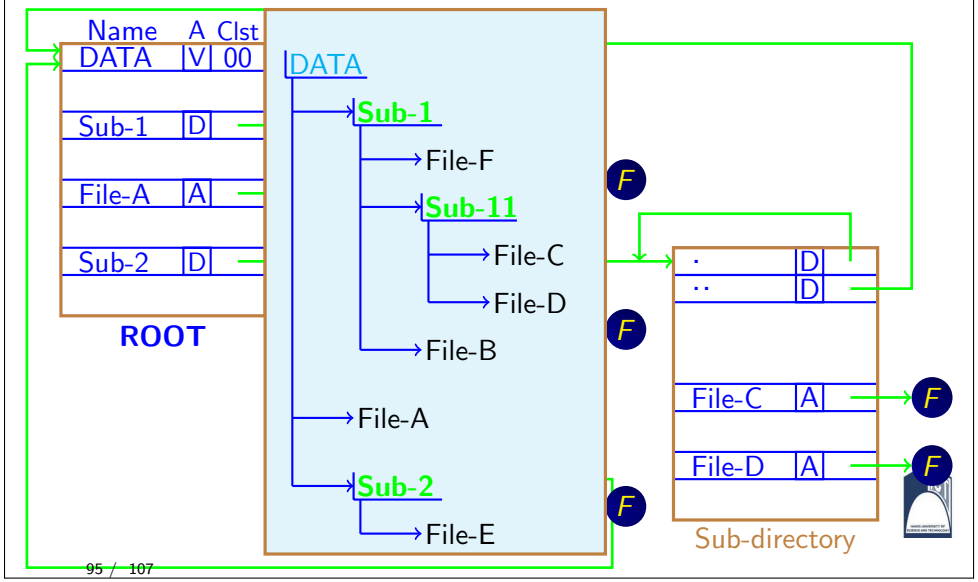
Cấu trúc một phần tử

Stt	Ofs	Size	Ý nghĩa
1	0	8B	Tên file
2	8	3B	Phần mở rộng
3	11	1B	Thuộc tính của file
4	12	10B	Không dùng với FAT12/FAT16. Sử dụng với FAT32
4.1	12	1B	Để dành
4.2	13	1B	Thời điểm tạo file, theo đơn vị 10ms
4.3	14	1W	Thời điểm tạo file (*giờ - phút - giây*)
4.4	16	1W	Ngày tạo file (*tạo bởi ứng dụng hoặc bởi copy sang*)
4.5	18	1W	Ngày truy nhập cuối
4.6	20	1W	Số hiệu cluster bắt đầu của file(*FAT32: Phần cao*)
5	22	1W	Thời gian cập nhật cuối cùng
6	24	1W	Ngày cập nhật cuối (*không y/cầu sau ngày tạo file*)
7	26	1W	Số hiệu cluster bắt đầu của file (*FAT32: Phần thấp*)
8	28	1DW	Kích thước tính bằng byte

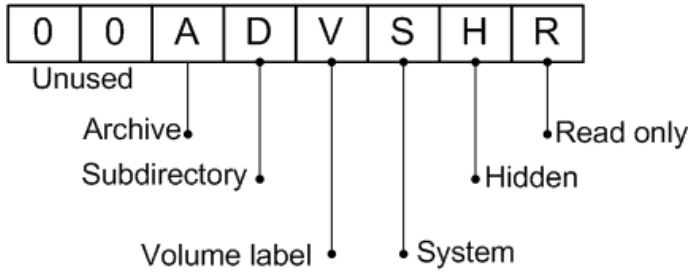
Cấu trúc một phần tử :Tên file

- Chuỗi ASCII chứa tên file. Các ký tự là chữ in
- Không chấp nhận khoảng trống ở giữa
  - Các câu lệnh copy, del,... không nhận biết tên có dấu trắng
- Nếu ít hơn 8 ký tự, được chèn các ký tự trống cho đủ 8
- Ký tự đầu có thể mang ý nghĩa đặc biệt
  - 00h: Phần tử đầu tiên của phần chưa dùng đến
  - E5h (ký tự "δ"): File tương ứng với phần tử này đã bị xóa.
  - 2Eh (ký tự "."): Đây là thư mục con
    - Trường số hiệu cluster bắt đầu chỉ đến chính nó
    - Cấu trúc như thư mục con giống như thư mục gốc: gồm các phần tử 32bytes
  - 2Eh2Eh (ký tự ".."): Đây là thư mục cha của thư mục hiện tại
    - Trường số hiệu cluster bắt đầu chỉ đến thư mục cha
    - Nếu cha là gốc, #cluster bắt đầu bằng zero (FAT12/16)
    - Thư mục con nằm trên phần Data, được quản lý như một file ⇒ File của các bản ghi file
    - FAT12/16: Thư mục gốc ở vị trí xác định; FAT32: Thư mục gốc cũng nằm trong phần data

Thư mục con



Cấu trúc một phần tử : Trường thuộc tính

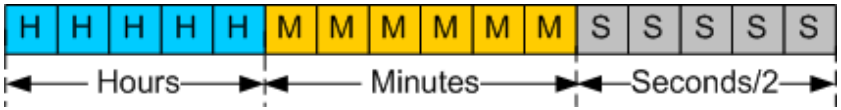


Ví dụ: Byte thuộc tính **0Fh**: | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

⇒ Có các thuộc tính Volume label+System+Hidden+Read only

Ghi chú: Giá trị byte thuộc tính **0x0F** không sử dụng trong *MS-DOS* ⇒ Dùng để đánh dấu là phần tử *Long File Name*

Cấu trúc một phần tử: Trường thời gian



Ví dụ: 15 giờ 34 phút 45 giây

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

Có giá trị : **7C56**

Cấu trúc một phần tử : Trường ngày tháng

Ví dụ: 17 tháng 5 năm 2011

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

Có giá trị : **3EB1**

Hệ thống Long File Name (LFN)

...
Phần tử LFN 3
Phần tử LFN 2
Phần tử LFN 1
Phần tử 8.3 (ttt∼n.xxx)

Ofs	Kt	Ý nghĩa
0	1B	Trường thứ tự.
1	5W	5 ký tự unicode đầu tiên
11	1B	Thuộc tính. Đánh dấu là phần tử *LFN*. Luôn có giá trị 0Fh
12	1B	Để dành (00)
13	1B	Checksum: Cho phép kiểm tra tên file dài có ứng với tên file 8.3?
14	6W	Các ký tự unicode 6,7,8,9,10,11
26	1W	Số hiệu cluster. Không dùng (0000)
28	1W	Ký tự unicode 12
30	1W	Ký tự unicode 13

Hệ thống Long File Name: Trường thứ tự

- Cho biết trật tự các phần tử LFN
  - Mỗi phần tử LFN chứa 13 ký tự Unicode
- Phần tử đầu tiên có giá trị trường thứ tự bằng 1
- Phần tử cuối sẽ dùng bít số 6 để đánh dấu
  - Chỉ dùng tối đa 20 phần tử
  - Sau ký tự cuối cùng là 0x00 0x00.
  - Các ký tự không sử dụng có giá trị 0xFF 0xFF
- Bít số 7 (0x80) cho biết phần tử tương ứng đã bị xóa
- Ví dụ file "*This is a very long file name.docx*"

Entry	Ord	Attr	Data
LFN 3	0x43	0x0F	ame.docx
LFN 2	0x02	0x0F	y long file n
LFN 1	0x01	0x0F	This is a ver
8.3 Name	THISIS∼1.DOC		

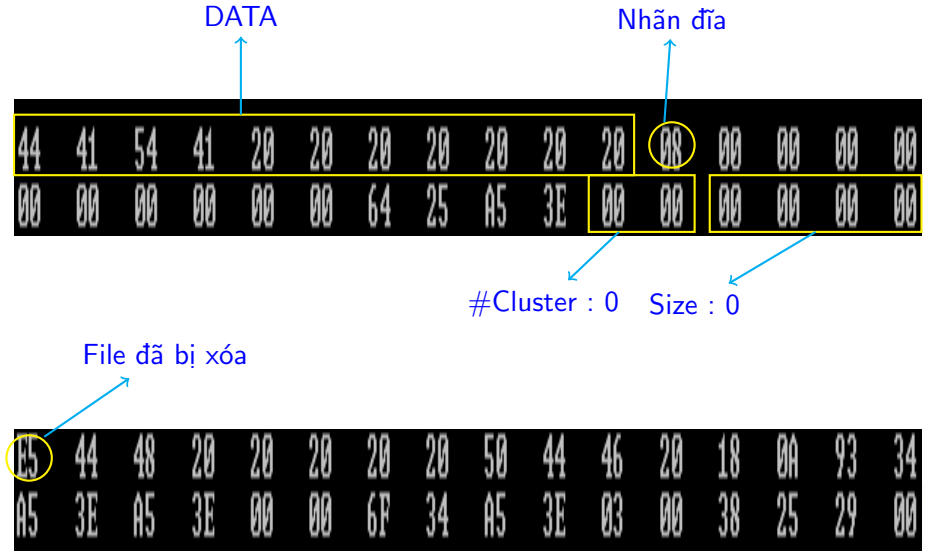
Ví dụ: Một sector của ROOT

```
44 41 54 41 20 20 20 20 20 20 20 08 00 00 00 00
00 00 00 00 00 00 64 25 A5 3E 00 00 00 00 00 00
E5 44 48 20 20 20 20 20 50 44 46 20 18 0A 93 34
A5 3E A5 3E 00 00 6F 34 A5 3E 03 00 38 25 29 00
41 45 00 78 00 65 00 6D 00 70 00 0F 00 EF 6C 00
65 00 73 00 00 00 FF FF FF FF 00 00 FF FF FF FF
45 58 45 4D 50 4C 45 53 20 20 20 10 00 C4 9B 34
A5 3E A5 3E 00 00 9C 34 A5 3E 96 14 00 00 00 00
42 72 00 2E 00 63 00 00 00 FF FF 0F 00 43 FF FF
FF FF FF FF FF FF FF FF FF 00 00 FF FF FF FF FF
01 52 00 65 00 61 00 64 00 42 00 0F 00 43 69 00
6F 00 73 00 53 00 65 00 63 00 00 00 74 00 6F 00
52 45 41 44 42 49 7E 31 43 20 20 20 00 A6 B2 4B
A5 3E A5 3E 00 00 76 5C 9A 3E 3F 2E D1 01 00 00
52 45 41 44 4D 42 52 20 43 20 20 00 3C 86 5B 00
A5 3E A5 3E 00 00 CF 79 A4 3E 40 2E BD 0A 00 00
41 54 00 65 00 6D 00 70 00 73 00 0F 00 F0 00 00
FF FF FF FF FF FF FF FF FF FF 00 00 FF FF FF FF
54 45 4D 50 53 20 20 20 20 20 20 10 00 11 9A 96
A5 3E A5 3E 00 00 9B 96 A5 3E 46 2E 00 00 00 00
42 A1 01 6E 00 67 00 20 00 34 00 0F 00 12 2E 00
70 00 64 00 66 00 00 00 FF FF 00 00 FF FF FF FF
01 42 00 E0 00 69 00 20 00 67 00 0F 00 12 69 00
A3 1E 6E 00 67 00 20 00 63 00 00 00 68 00 B0 01
42 41 49 47 49 4E 7E 31 50 44 46 20 00 0A 93 34
A5 3E A5 3E 00 00 6F 34 A5 3E 03 00 38 25 29 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

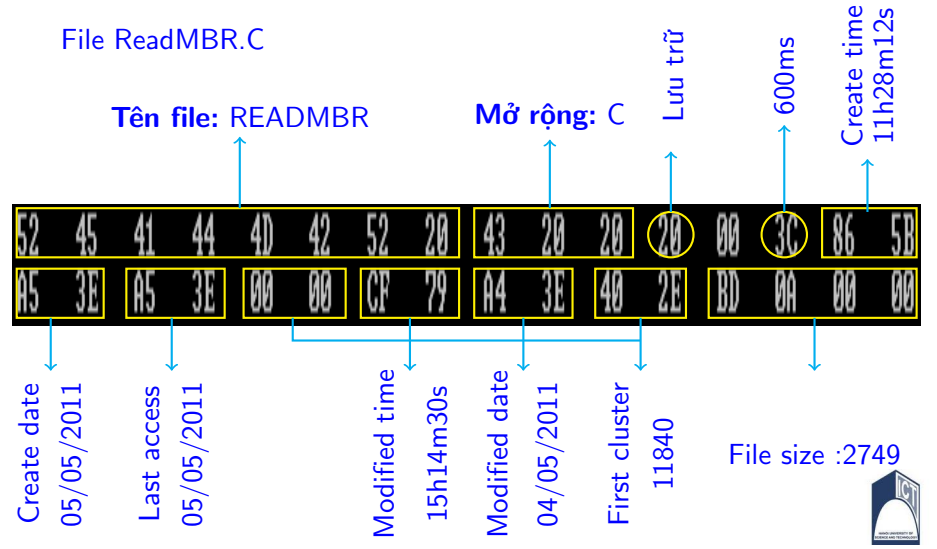

Ví dụ: Nội dụng của ROOT

```
F:\>DIR
 Volume in drive F is DATA
 Volume Serial Number is DC27-F353

 Directory of F:\

05/05/2011  06:36 AM    <DIR>          Exemples
04/26/2011  11:35 AM               465 ReadBiosSector.c
05/04/2011  03:14 PM             2,749 READMBR.C
05/05/2011  06:52 PM    <DIR>          Temps
05/05/2011  06:35 AM         2,696,504 Bài gi?ng chuong 4.pdf
               3 File(s)      2,699,718 bytes
               2 Dir(s)      14,247,424 bytes free

F:\>_
```


Giải mã ROOT 1

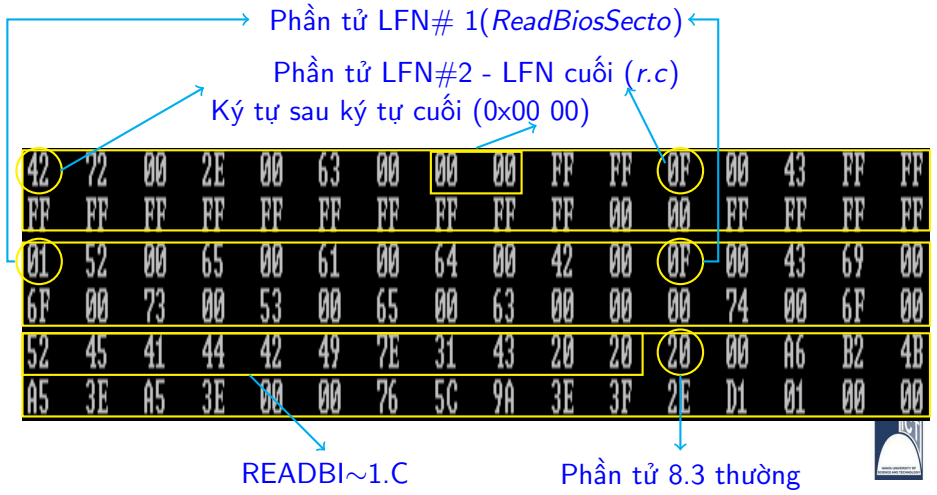


Giải mã ROOT 2



Giải mã ROOT 3

File: **ReadBiosSector.c**



Bài tập

1. Viết chương trình Diskedit
   - Cho phép xem (và sửa chữa) từng sector của một đĩa cứng.
   - Các sector được hiện thị dưới cả 2 dạng: Hexa và ASCII
2. Viết chương trình liệt kê tất cả các phân vùng của ổ đĩa cứng.
   - Nếu phân vùng sử dụng hệ thống file FAT32 hoặc NTFS, đưa ra các thông tin tương ứng
3. Viết chương trình đưa ra nội dung của thư mục gốc của đĩa cứng sử dụng FAT32
   - Chỉ sử dụng thủ tục đọc sector trên đĩa
4. Nghiên cứu cách tổ chức của các hệ thống file NTFS, EXT3
5. Xây dựng một hệ thống file trên một đĩa ảo

Kết luận