# 一、文件恢复

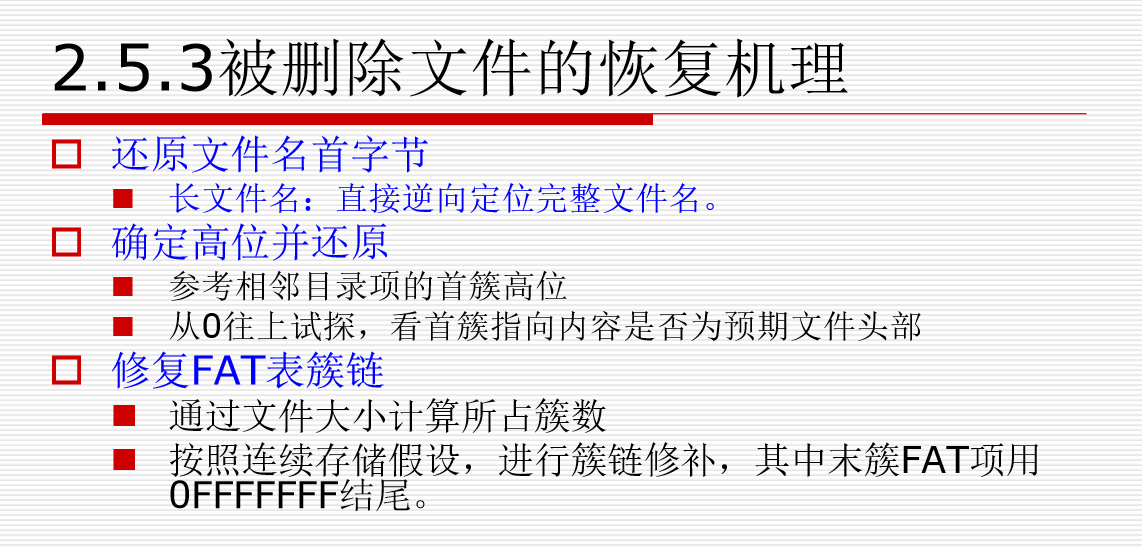
1.通过MBR🡪DBR🡪FAT个数找到根目录，然后一层层索引找到被删的FDT文件记录项，恢复文件记录项

具体：BPB结构存储了和该分区有关的重要信息，里面有:

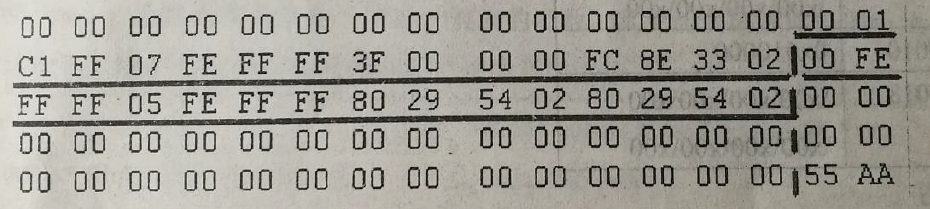
打开磁盘首先进入的是DBR，此时FILE\_BEGIN指针为零，是指对DBR的开始位置而言偏移0，保留扇区数的作用是可以通过它，获得FAT 1相对于FILE\_BEGIN的偏移。FAT的个数一般是2个，在知道每个FAT占用的扇区数之后，可以通过该字段，获得根目录相对**FILE\_BEGIN(其实就是分区的开头，如果是Ｕ盘，就是绝对地址)**的偏移。根目录的相对开始扇区号=保留扇区数+FAT个数\*每个FAT的扇区数。如果要计算绝对地址，则需要用到Hidden Sectors，它指的是该分区DBR之前的扇区数。在本实验中，使用相对地址就可以。

2.然后修复FDT项目，修复簇链

3.长目录就不考虑了，简答题......



# 二、磁盘计算(不同取法，最终结果会不一样，没有标准答案)



①精确计算：主分区大小0x2338EFC，起始位置0x3F也就是第63扇区，结束位置(0x3F+0x2338EFC-0x1)号扇区，相当于0x2338EFC右移21位，也就是约17.6G，17.61120414733887GB

十进制强行算：上面那个十进制是36933372，如果其它算法，按照十进制算是18.466686GB

**手酸**(取成MB的形式)：35\*220\*512/1024/1024/1024，按照题目定义1K=1000，也就是35\*106\*500/1000/1000/1000=35/2=17.5GB

精确些手算：右移10位，得到0x8CE3=36067，所以是8CE3\*210，按照1K=1000也就是36067\*1000，然后继续计算，得到36067\*1000\*500/1000/1000/1000=18.0GB

②精确计算：因为标志位是0x5所以是扩展分区，扩展分区大小为0x2542980，十进制是39070080个扇区，也就是18.6GB

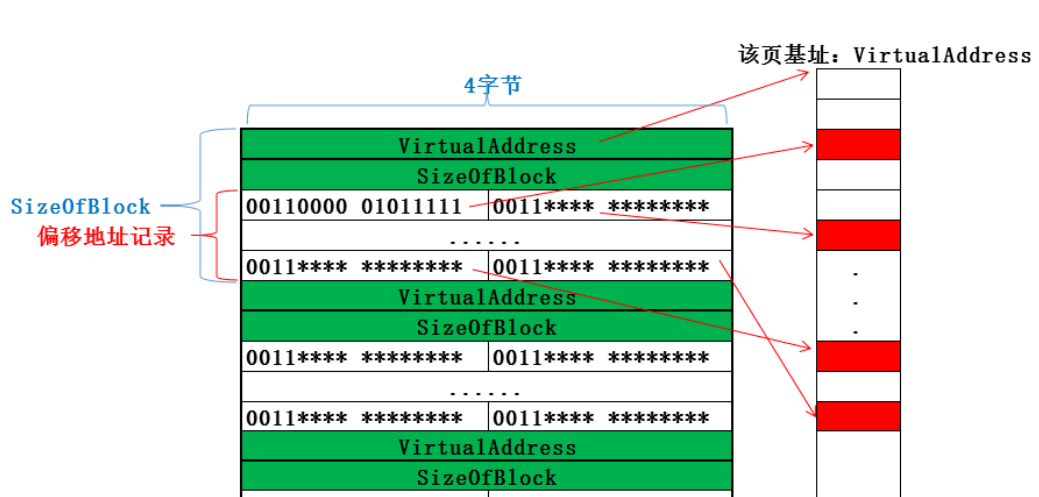
**手算**(取成MB的形式)：相当于约37\*220，37\*106\*500/1000/1000/1000=18.5GB

# 三、PE文件重定位节

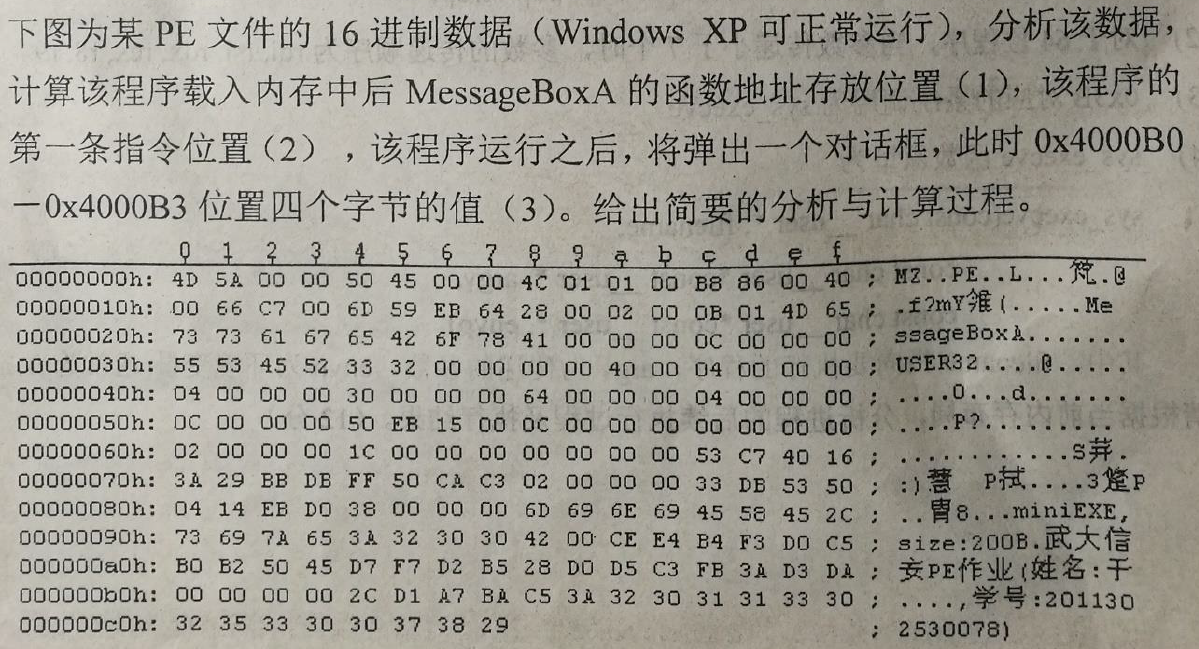
链接器生成一个PE文件时，它会假设程序被装入时使用的默认ImageBase基地址（VC默认exe基地址00400000h，dll基地址10000000h），并且会把代码中所有指令中用到的地址都使用默认的基地址（例如 程序代码中 push 10001000，就是把10000000h当做了基地址，把push 10001000写入到文件中）。如果一个exe程序中一个dll装载时的地址与其它dll地址发生冲突（因为windows程序是虚拟地址空间，exe一般不会有地址冲突，加载dll时可能会有地址冲突），就需要修改代码中的地址，如push 10001000，call 10002000等。这时就需要用进行基址重定位。而基址重定位表中存放了，如果默认地址被改，需要修改的代码的地址。在PE文件中，基址重定位表一般放在一个单独的 ".reloc" 区，可以通过IMAGE\_OPTIONAL\_HEADER 中 的DataDirectory[5] 查看 基址重定位表 的RVA。

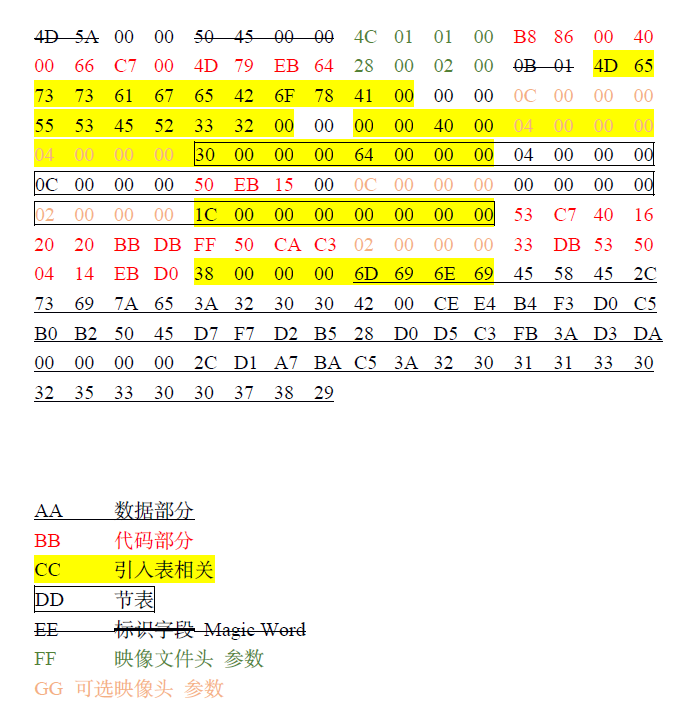
例如：可以发现MyMessageBox 这个函数，看看它的代码中的push 10006040 ， push 10006030 中的地址是指向字符串的。如果一个程序在加载Demo.dll时因为Demo.dll 默认的地址被占用了，而使用其它的基地址，例如使用20000000h作为基地址，Demo.dll就从20000000h开始装载。这样字符串“Demo”和“Hello World！” 就不是在10006040h跟10006030h中了，这时就需要把push 10006040 ， push 10006030改成push 20006040 ， push 20006030 。

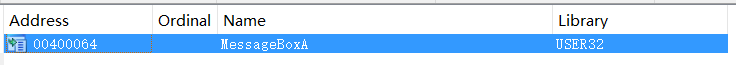
重定位表结构：其中 VirtualAddress 表示 这一组地址的起始RVA。SizeOfBlock表示当前这个IMAGE\_BASE\_RELOCATION 结构的大小。该结构体有两个成员：一个是地址，一个是大小。如下图所示：一个重定位表由多个大小SizeOfBlock的Block组成，（不同块的SizeOfBlock大小不一）。每一个块记录了1000H即4KB大小的内存中需要重定位信息的地址（一页大小），这些地址以VirtualAdress为该页的基址，偏移地址占两个字节（1000H最多需要12bit即可：0~FFFH）。所以两个字节的低12位为偏移地址，而高4位就是一个标记，当此标记为0011（3）时低12为才有效，否则该2个字节可能是为了对齐而产生的，并且为对齐而产生的字节其值全为0。

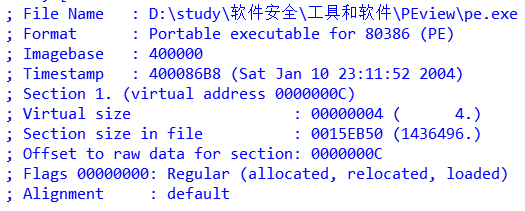


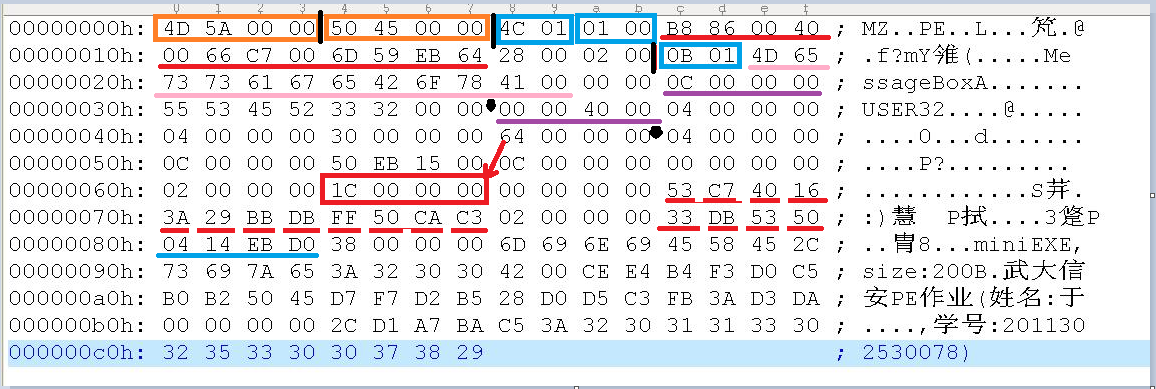
# 四、最小PE文件计算题











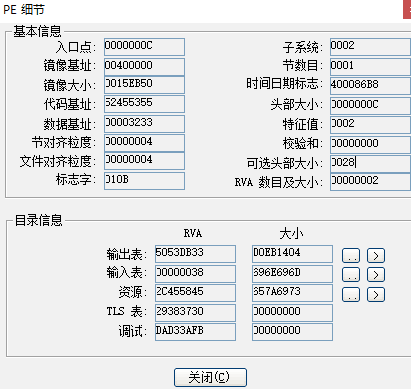
很多地方复用了而已，结构没有变。

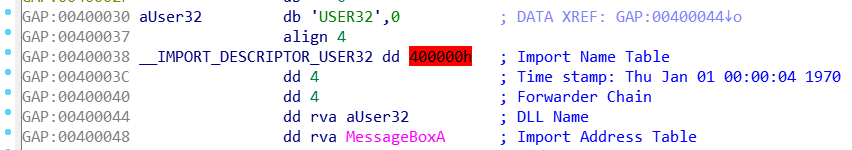
0x00-0x03是PE，然后是5045，然后的0x14字节是映像文件头，只不过里面出了开头4字节是原来的，后面的塞了代码。

IDT表从0x38箭头开始。

复用从0x1C开始，一直是可选映像头，但是里面是有其它东西

0xB0处是指向0x70的指针....





将其控制流整理一下：

seg000:0000000C ; ---------------------------------------------------------------------------

|  |  |  |
| --- | --- | --- |
| seg000:0000000C | mov | eax, 400086h |
| seg000:00000011 | mov | word ptr [eax], 794Dh |
| seg000:00000016 | jmp | short loc\_7C |

seg000:0000007C ; --------------------------------------------------------------------------- seg000:0000007C

seg000:0000007C loc\_7C: CODE XREF: seg000:00000016↑j

seg000:0000007C xor ebx, ebx

seg000:0000007E push ebx

seg000:0000007F push eax

seg000:00000080 add al, 14h

seg000:00000082 jmp short loc\_54

seg000:00000054 ; --------------------------------------------------------------------------- seg000:00000054

seg000:00000054 loc\_54: ; CODE XREF: seg000:00000082↓j

seg000:00000054 push eax

seg000:00000055 jmp short loc\_6C

seg000:0000006C ; --------------------------------------------------------------------------- seg000:0000006C

seg000:0000006C loc\_6C: ; CODE XREF: seg000:00000055↑j

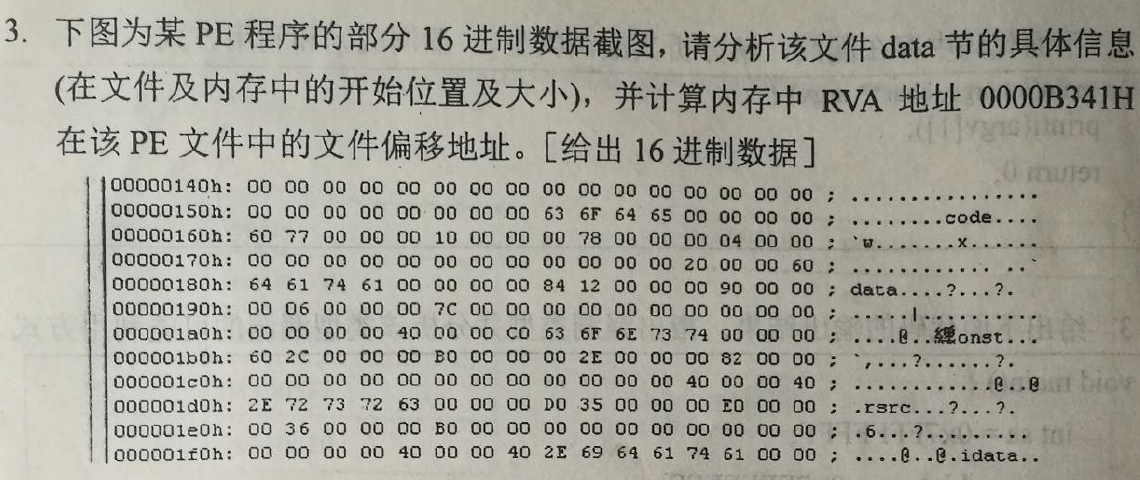
seg000:0000006C push ebx

seg000:0000006D mov dword ptr [eax+16h], 0DBBB2020h

seg000:00000074 call dword ptr [eax-36h]

seg000:00000077 retn

# 五、PE文件计算题



data节开始于0x180，起始RVA是0x9000，起始pFile是0x7C00

文件偏移计算：0xB341-(0x9000-0x7C00)=0x9F41

# 六、整型溢出

void main()

{

int sa=0x7FFFFFFF;

unsigned int ua=0x7FFFFFFF;

sa=sa+2;

ua=ua+2;

printf("sa: %d \t ua: %d\n",sa,ua);

if(sa<1)

printf("sa is overflow");

if(ua<1)

printf("ua is overflow");

}

2^32= 4294967296

结果：

sa: -2147483647 ua: -2147483647

sa is overflow

# 七、信息泄露

int main()

{

int i=1,j=2,k=3;

char buf[]="test";

printf("%s %d %d %d\n",buf,i,j);

return 0;

}

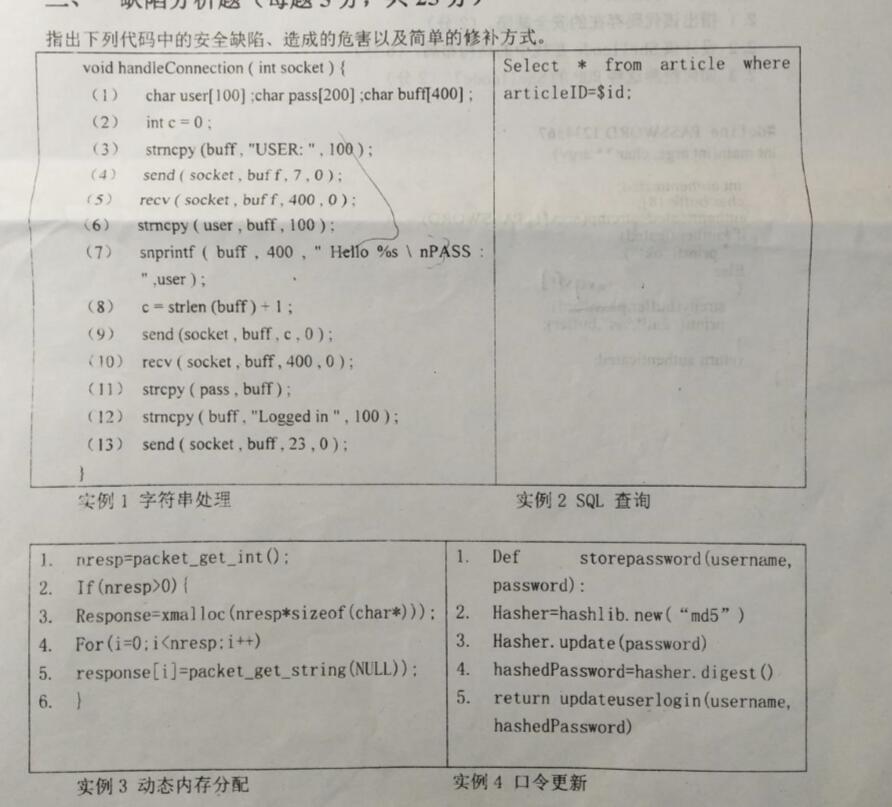
结果：

test 1 2 1953719668

也就是“test”的十六进制74736574h【因为只有4个字母，直接压字母进栈】

explorer.exe是Windows程序管理器或者文件资源管理器，它用于管理Windows图形壳，包括桌面和文件管理，删除该程序会导致Windows图形界面无法使用。谈一下功能

API的：不同操作系统可能版本不同、PEB不同



xmalloc，分配内存失败会直接退出，换成malloc