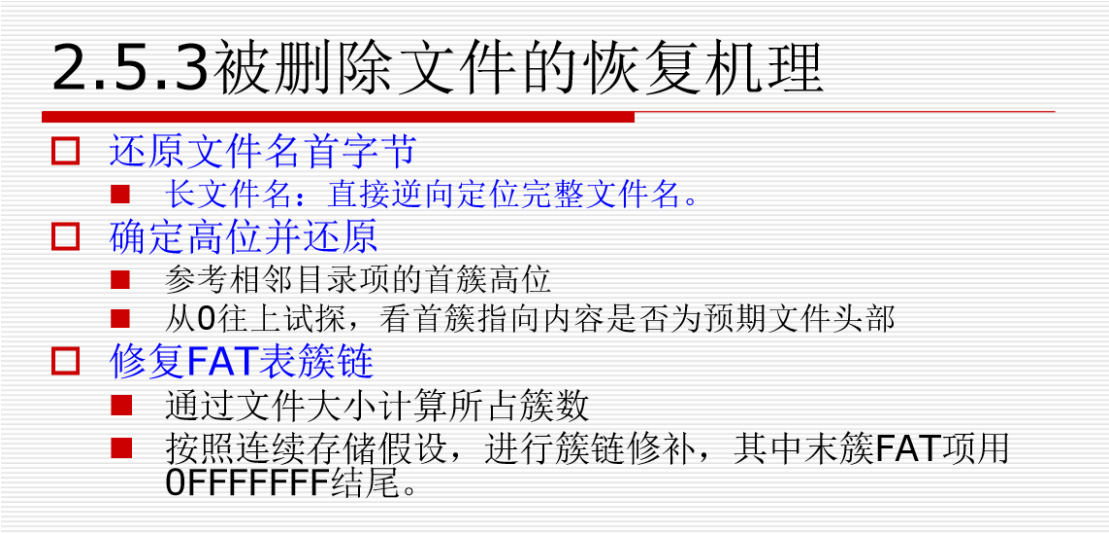
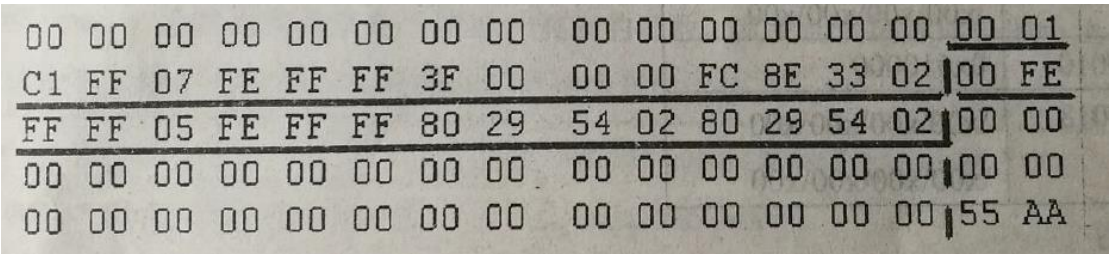
一、文件恢复

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*2^{20}*512/1024/1024/1024，按照题目定义 1K=1000，也就是

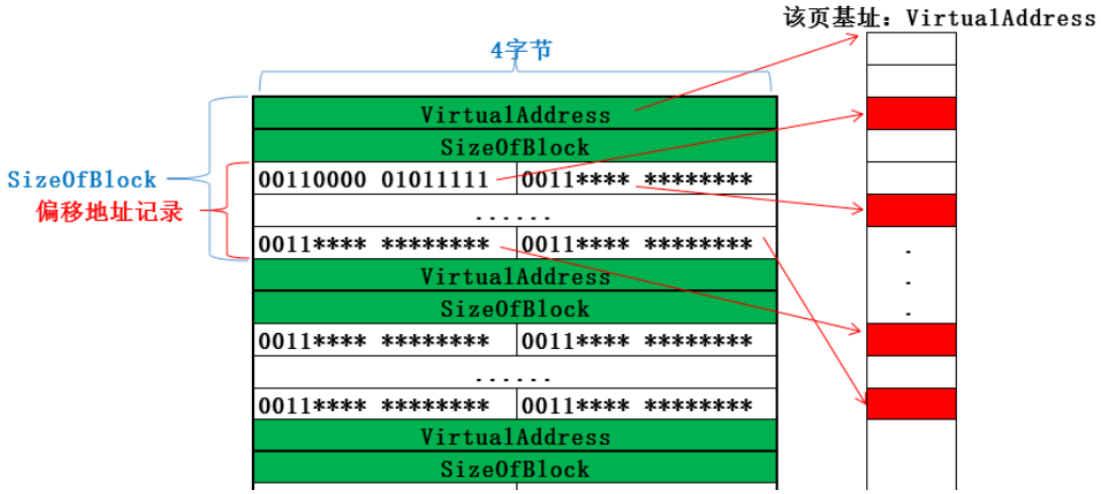
$35*10^6*500/1000/1000/1000=35/2=17.5GB$
精确些手算：右移 10 位，得到 0x8CE3=36067，所以是 $8CE3*2^{10}$，按照 1K=1000 也就是 36067*1000，然后继续计算，得到 36067*1000*500/1000/1000/1000=18.0GB

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 $37*2^{20}$，$37*10^6*500/1000/1000/1000=18.5GB$

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

下图为某 PE 文件的 16 进制数据（Windows XP 可正常运行），分析该数据，计算该程序载入内存中后 MessageBoxA 的函数地址存放位置（1），该程序的第一条指令位置（2），该程序运行之后，将弹出一个对话框，此时 0x4000B0—0x4000B3 位置四个字节的值（3）。给出简要的分析与计算过程。

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 4D 5A 00 00 50 45 00 00 4C 01 01 00 B8 86 00 40 ; MZ..PE..L...兗.@
00000010h: 00 66 C7 00 6D 59 EB 64 28 00 02 00 0B 01 4D 65 ; .f?mY雅(.....Me
00000020h: 73 73 61 67 65 42 6F 78 41 00 00 00 0C 00 00 00 ; ssageBoxA.......
00000030h: 55 53 45 52 33 32 00 00 00 00 40 00 04 00 00 00 ; USER32....@.....
00000040h: 04 00 00 00 30 00 00 00 64 00 00 00 04 00 00 00 ; ....0...d.......
00000050h: 0C 00 00 00 50 EB 15 00 0C 00 00 00 00 00 00 00 ; ....P?.........
00000060h: 02 00 00 00 1C 00 00 00 00 00 00 00 53 C7 40 16 ; ............S芽.
00000070h: 3A 29 BB DB FF 50 CA C3 02 00 00 00 33 DB 53 50 ; :)簪  P拭....3篷P
00000080h: 04 14 EB D0 38 00 00 00 6D 69 6E 69 45 58 45 2C ; ..胄8...miniEXE,
00000090h: 73 69 7A 65 3A 32 30 30 42 00 CE E4 B4 F3 D0 C5 ; size:200B.武大信
000000a0h: B0 B2 50 45 D7 F7 D2 B5 28 D0 D5 C3 FB 3A D3 DA ; 安PE作业(姓名:于
000000b0h: 00 00 00 00 2C D1 A7 BA C5 3A 32 30 31 31 33 30 ; ....,学号:201130
000000c0h: 32 35 33 30 30 37 38 29                         ; 2530078)
```

~~4D 5A~~ 00 00 ~~50 45 00 00~~ 4C 01 01 00 B8 86 00 40
00 66 C7 00 4D 79 EB 64 28 00 02 00 ~~0B 01~~ 4D 65
73 73 61 67 65 42 6F 78 41 00 00 00 0C 00 00 00
55 53 45 52 33 32 00 00 00 00 40 00 04 00 00 00
04 00 00 00 30 00 00 00 64 00 00 00 04 00 00 00
0C 00 00 00 50 EB 15 00 0C 00 00 00 00 00 00 00
02 00 00 00 1C 00 00 00 00 00 00 00 53 C7 40 16
20 20 BB DB FF 50 CA C3 02 00 00 00 33 DB 53 50
04 14 EB D0 38 00 00 00 6D 69 6E 69 45 58 45 2C
73 69 7A 65 3A 32 30 30 42 00 CE E4 B4 F3 D0 C5
B0 B2 50 45 D7 F7 D2 B5 28 D0 D5 C3 FB 3A D3 DA
00 00 00 00 2C D1 A7 BA C5 3A 32 30 31 31 33 30
32 35 33 30 30 37 38 29

AA 数据部分
BB 代码部分
CC 引入表相关
DD 节表
~~EE 标识字段 Magic Word~~
FF 映像文件头 参数
GG 可选映像头 参数

Address	Ordinal	Name	Library
00400064		MessageBoxA	USER32

```
; File Name   : D:\study\软件安全\工具和软件\PEview\pe.exe
; Format      : Portable executable for 80386 (PE)
; Imagebase   : 400000
; Timestamp   : 400086B8 (Sat Jan 10 23:11:52 2004)
; Section 1. (virtual address 0000000C)
; Virtual size                  : 00000004 (      4.)
; Section size in file          : 0015EB50 (1436496.)
; Offset to raw data for section: 0000000C
; Flags 00000000: Regular (allocated, relocated, loaded)
; Alignment     : default
```

```
00000000h: 4D 5A 00 00 50 45 00 00 4C 01 01 00 B8 86 00 40 ; MZ..PE..L...笕.@
00000010h: 00 66 C7 00 6D 59 EB 64 28 00 02 00 0B 01 4D 65 ; .f?mY雏(.....Me
00000020h: 73 73 61 67 65 42 6F 78 41 00 00 00 0C 00 00 00 ; ssageBoxA.......
00000030h: 55 53 45 52 33 32 00 00 00 00 40 00 04 00 00 00 ; USER32....@.....
00000040h: 04 00 00 00 30 00 00 00 64 00 00 00 04 00 00 00 ; ....0...d.......
00000050h: 0C 00 00 00 50 EB 15 00 0C 00 00 00 00 00 00 00 ; ....P?.........
00000060h: 02 00 00 00 1C 00 00 00 00 00 00 00 53 C7 40 16 ; ............S芾.
00000070h: 3A 29 BB DB FF 50 CA C3 02 00 00 00 33 DB 53 50 ; :)慧  P拭....3篗P
00000080h: 04 14 EB D0 38 00 00 00 6D 69 6E 69 45 58 45 2C ; ..胄8...miniEXE,
00000090h: 73 69 7A 65 3A 32 30 30 42 00 CE E4 B4 F3 D0 C5 ; size:200B.武大信
000000a0h: B0 B2 50 45 D7 F7 D2 B5 28 D0 D5 C3 FB 3A D3 DA ; 安PE作业(姓名:于
000000b0h: 00 00 00 00 2C D1 A7 BA C5 3A 32 30 31 31 33 30 ; ....,学号:201130
000000c0h: 32 35 33 30 30 37 38 29                         ; 2530078)
```

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....


```
GAP:00400030 aUser32          db 'USER32',0           ; DATA XREF: GAP:00400044↓o
GAP:00400037                  align 4
GAP:00400038 __IMPORT_DESCRIPTOR_USER32 dd 400000h    ; Import Name Table
GAP:0040003C                  dd 4                    ; Time stamp: Thu Jan 01 00:00:04 1970
GAP:00400040                  dd 4                    ; Forwarder Chain
GAP:00400044                  dd rva aUser32          ; DLL Name
GAP:00400048                  dd rva MessageBoxA      ; Import Address Table
```

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

3. 下图为某 PE 程序的部分 16 进制数据截图，请分析该文件 data 节的具体信息(在文件及内存中的开始位置及大小)，并计算内存中 RVA 地址 0000B341H 在该 PE 文件中的文件偏移地址。[给出 16 进制数据]

```
00000140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000150h: 00 00 00 00 00 00 00 00 63 6F 64 65 00 00 00 00 ; ........code....
00000160h: 60 77 00 00 00 10 00 00 00 78 00 00 00 04 00 00 ; `w.......x......
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 ; ............ ..`
00000180h: 64 61 74 61 00 00 00 00 84 12 00 00 00 90 00 00 ; data....?...?.
00000190h: 00 06 00 00 00 7C 00 00 00 00 00 00 00 00 00 00 ; .....|..........
000001a0h: 00 00 00 00 40 00 00 C0 63 6F 6E 73 74 00 00 00 ; ....@..繧onst...
000001b0h: 60 2C 00 00 00 B0 00 00 00 2E 00 00 00 82 00 00 ; `,...?......?.
000001c0h: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 ; ............@..@
000001d0h: 2E 72 73 72 63 00 00 00 D0 35 00 00 00 E0 00 00 ; .rsrc...?...?.
000001e0h: 00 36 00 00 00 B0 00 00 00 00 00 00 00 00 00 00 ; .6...?.........
000001f0h: 00 00 00 00 40 00 00 40 2E 69 64 61 74 61 00 00 ; ....@..@.idata..
```

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;
}
```

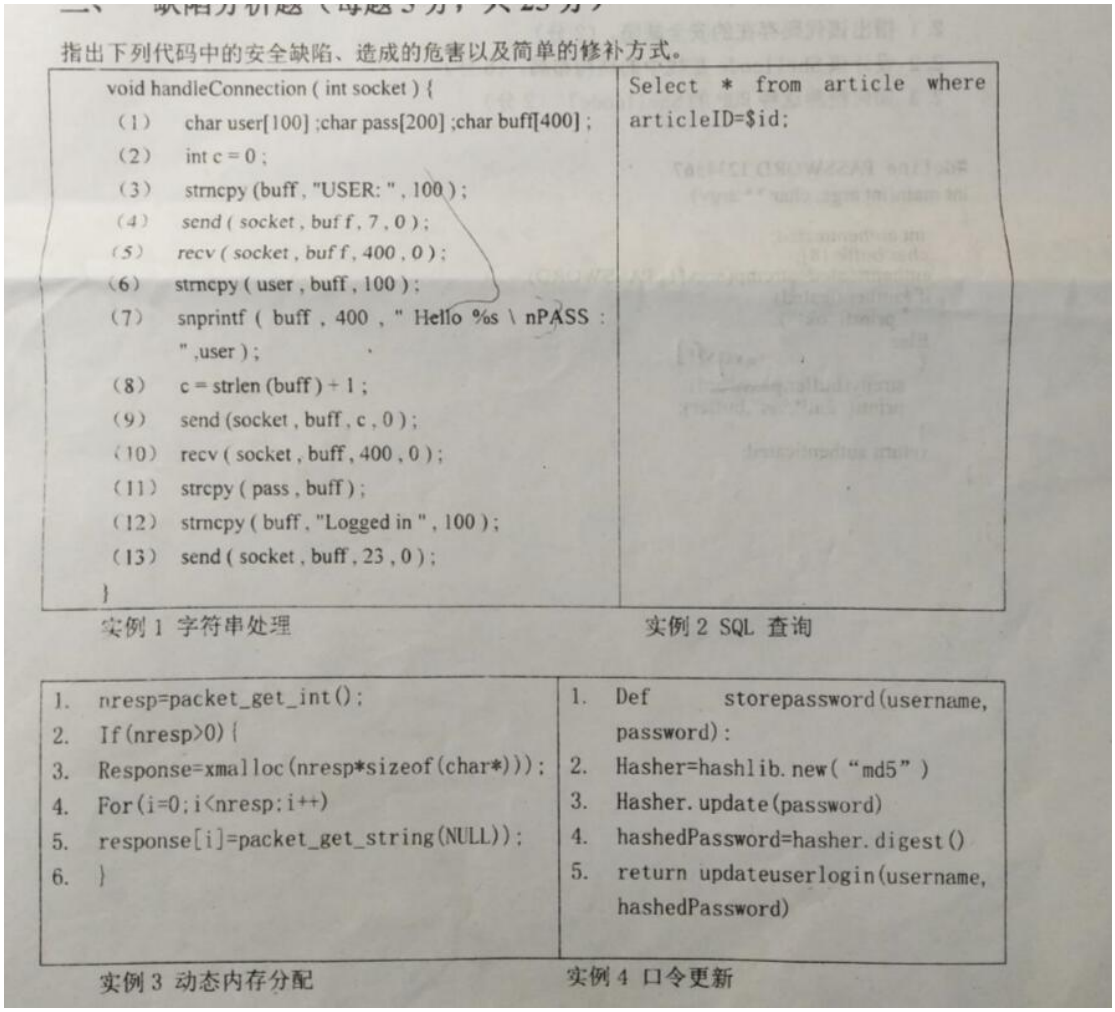
结果：

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



指出下列代码中的安全缺陷、造成的危害以及简单的修补方式。

```
void handleConnection ( int socket ) {
(1)  char user[100] ;char pass[200] ;char buff[400] ;
(2)  int c = 0 ;
(3)  strncpy (buff , "USER: " , 100 ) ;
(4)  send ( socket , buff , 7 , 0 ) ;
(5)  recv ( socket , buff , 400 , 0 ) ;
(6)  strncpy ( user , buff , 100 ) ;
(7)  snprintf ( buff , 400 , " Hello %s \ nPASS :
     " ,user ) ;
(8)  c = strlen (buff ) + 1 ;
(9)  send (socket , buff , c , 0 ) ;
(10) recv ( socket , buff , 400 , 0 ) ;
(11) strcpy ( pass , buff ) ;
(12) strncpy ( buff , "Logged in " , 100 ) ;
(13) send ( socket , buff , 23 , 0 ) ;
}
```

实例 1 字符串处理

```
Select * from article where
articleID=$id;
```

实例 2 SQL 查询

```
1. nresp=packet_get_int();
2. If(nresp>0){
3. Response=xmalloc(nresp*sizeof(char*)));
4. For(i=0;i<nresp;i++)
5. response[i]=packet_get_string(NULL));
6. }
```

实例 3 动态内存分配

```
1. Def storepassword(username,
   password):
2. Hasher=hashlib.new(“md5”)
3. Hasher.update(password)
4. hashedPassword=hasher.digest()
5. return updateuserlogin(username,
   hashedPassword)
```

实例 4 口令更新

xmalloc，分配内存失败会直接退出，换成 malloc