汇编语言与逆向工程

北京邮电大学
付俊松

第六章 异常处理

- 一. Windows异常处理机制
- 二. SEH结构化异常处理
- 三. SEH反调原理
- 四. 逆向分析

（1）Windows异常处理机制

- 1.异常
- 2.异常处理机制
- 3.Windows异常处理流程

(1) Windows异常处理机制 — 异常

1.异常

- 异常（exception）是程序运行过程中发生的程序本身可以处理的错误，也是程序必须处理、无法忽略的错误。
  - 有些是因为程序本身的逻辑错误，如非法除零、非法内存读写等；有些是用户操作导致程序错误，如用户误删文件导致文件无法找到等。
  - 异常就是程序运行过程中由于种种原因导致的意外情况。
  - 程序通过抛出异常的形式将这些意外情况告诉上级调用者，系统会强制调用者对异常进行处理。

(1) Windows异常处理机制 — 异常

– 微软提供的**winnt.h**文件中对程序运行中可能遇到的多种异常进行了定义（**VC++6.0 Debug**版本）

```
#define STATUS_GUARD_PAGE_VIOLATION      ((DWORD   )0x80000001L)
#define STATUS_DATATYPE_MISALIGNMENT     ((DWORD   )0x80000002L)
#define STATUS_BREAKPOINT                ((DWORD   )0x80000003L)
#define STATUS_SINGLE_STEP               ((DWORD   )0x80000004L)
#define STATUS_ACCESS_VIOLATION          ((DWORD   )0xC0000005L)
#define STATUS_IN_PAGE_ERROR             ((DWORD   )0xC0000006L)
#define STATUS_INVALID_HANDLE            ((DWORD   )0xC0000008L)
#define STATUS_NO_MEMORY                 ((DWORD   )0xC0000017L)
#define STATUS_ILLEGAL_INSTRUCTION       ((DWORD   )0xC000001DL)
#define STATUS_NONCONTINUABLE_EXCEPTION  ((DWORD   )0xC0000025L)
#define STATUS_INVALID_DISPOSITION       ((DWORD   )0xC0000026L)
#define STATUS_ARRAY_BOUNDS_EXCEEDED     ((DWORD   )0xC000008CL)
#define STATUS_FLOAT_DENORMAL_OPERAND    ((DWORD   )0xC000008DL)
#define STATUS_FLOAT_DIVIDE_BY_ZERO      ((DWORD   )0xC000008EL)
#define STATUS_FLOAT_INEXACT_RESULT      ((DWORD   )0xC000008FL)
#define STATUS_FLOAT_INVALID_OPERATION   ((DWORD   )0xC0000090L)
#define STATUS_FLOAT_OVERFLOW            ((DWORD   )0xC0000091L)
#define STATUS_FLOAT_STACK_CHECK         ((DWORD   )0xC0000092L)
#define STATUS_FLOAT_UNDERFLOW           ((DWORD   )0xC0000093L)
#define STATUS_INTEGER_DIVIDE_BY_ZERO    ((DWORD   )0xC0000094L)
#define STATUS_INTEGER_OVERFLOW          ((DWORD   )0xC0000095L)
#define STATUS_PRIVILEGED_INSTRUCTION    ((DWORD   )0xC0000096L)

#define STATUS_STACK_OVERFLOW            ((DWORD   )0xC00000FDL)
```

(1) Windows异常处理机制 — 异常

– 五种最具代表性的异常，在程序运行过程中最为常见

异常种类	含义
EXCEPTION_ACCESS_VIOLATION(C0000005)	试图访问不存在或无访问权限的内存区域
EXCEPTION_BREAKPOINT(80000003)	在运行代码中设置断点后，CPU尝试执行该地址处的指令时就会发生断点异常
EXCEPTION_ILLEGAL_INSTRUCTION(C000001D)	CPU遇到无法解析的命令
EXCEPTION_INT_DIVIDE_BY_ZERO(C0000094)	整数除法运算中若分母为零则引发除零异常
EXCEPTION_SINGLE_STEP(80000004)	CPU在单步工作模式下，每执行一条指令就会引发一次单步异常

（1）Windows异常处理机制 — 异常处理机制

- 2.异常处理机制
  - 对异常的处理称为异常处理（exceptional handling）。
  - 异常处理通过处理程序运行时可能出现的异常情况，尽可能的使程序不受异常影响稳健运行。
  - 一个异常结构完备的系统不会有运行时错误。

（1）Windows异常处理机制 — 异常处理机制

- 从分离代码的角度上看，异常处理机制也可以看做一种分支处理机制。
- 将异常看做程序运行中不常见的分支情况，异常处理就是对这些不常见的分支情况的处理
- 操作系统引进了异常处理机制，将这些不常见的分支归为异常，进行统一的异常处理，便于程序员在编写程序时将注意力集中于正常情况处理。

（1）Windows异常处理机制 — 异常处理机制

- 尽管异常处理机制可以理解为一种分支机制，但有些异常处理不可以由if/else条件结构替代。
- 因为有些异常是可预测的，有些异常是不可预测的
  - 像除零异常这类可预测的异常可以使用条件结构进行判断，但像文件读写、序列化对象等不可预测的异常就无法用条件结构进行控制，只能使用异常处理。
  - 条件判断是提前进行的，不一定能准确地发现异常，而对于异常的判断是实时的，是与代码运行同时的。
- 在windows操作系统中，异常处理机制由SEH来实现

（1）Windows异常处理机制 — Windows异常处理流程

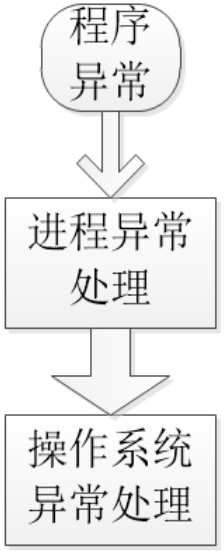
3.Windows异常处理流程

- 程序运行分为正常运行与调试运行两种情况。
- 在不同的运行情况下，操作系统对程序的异常处理流程也有所不同。

（1）Windows异常处理机制 — Windows异常处理流程

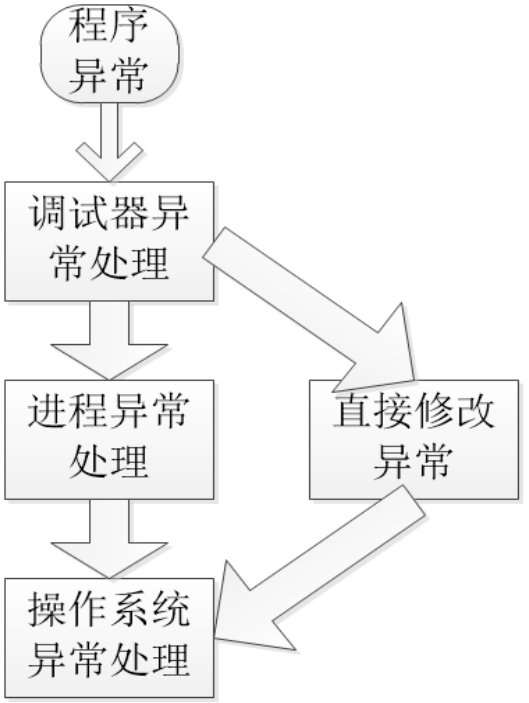
– 在程序正常运行的情况下，当异常发生时

- 操作系统会先将异常抛给进程处理，进程代码中如果存在具体的异常处理代码，则能顺利处理异常继续运行。
- 如果没有,则操作系统启动默认异常处理，终止进程运行



(1) Windows异常处理机制 — Windows异常处理流程

- 在调试运行的情况下，当异常发生时
  - 操作系统会先把异常抛给调试器进程，由调试人员进一步选择异常处理的方式。
  - 调试者在使用调试器处理被调程序异常时有两种方法
    - 直接修改代码、寄存器、内存来修改异常
    - 将异常抛给被调试程序处理
  - 如果这两种方法无法处理异常，则操作系统会使用默认异常处理机制进行处理，终止被调试程序，同时结束调试。



第六章 异常处理

- 一. Windows异常处理机制
- 二. SEH结构化异常处理
- 三. SEH反调原理
- 四. 逆向分析

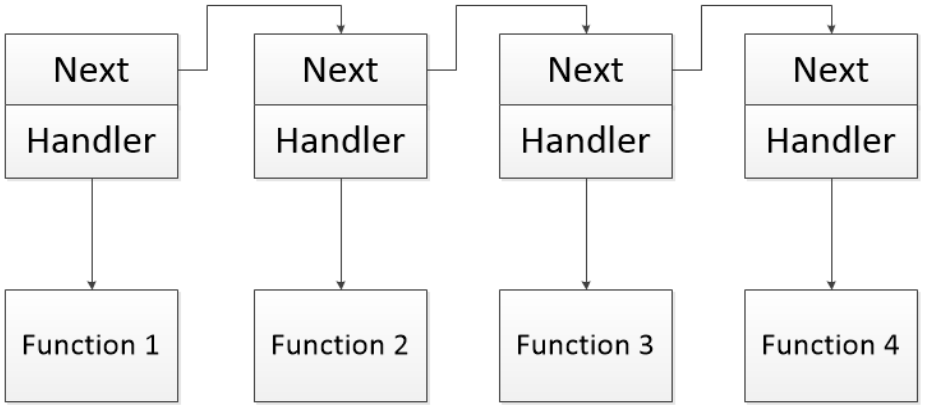
(2) SEH结构化异常处理

- 1. SEH链
- 2. SEH的应用原理
- 3. 在程序中添加SEH

(2) SEH结构化异常处理 — SEH链

1. SEH链

- SEH（Structured Exception Handling，结构化异常处理）是windows操作系统默认的异常处理机制，在程序源码中使用__try、__except关键字来具体实现。
- 从数据结构上来看，SEH以链表的形式存在，称为SEH链



(2) SEH结构化异常处理 — SEH链

- SEH链的节点是一个_EXCEPTION_REGISTRATION_RECORD结构体，称为异常处理器（有时异常处理器指的是异常处理函数）。
- 一个异常处理器有两个功能：
  - 提供用于处理异常的代码
  - 指明下一个异常处理器的位置

(2) SEH结构化异常处理 — SEH链

- _EXCEPTION_REGISTRATION_RECORD结构体有Next和Handler两个成员
  - Next成员是一个结构体指针，指向下一个_EXCEPTION_REGISTRATION_RECORD结构体，也就是下一个异常处理器的地址，如果Next的值为FFFFFFFF，则表示SEH链到此结束
  - Handler成员是一个函数指针，指向异常处理函数。异常处理函数是一个回调函数，由系统调用

```
Typedef struct _EXCEPTION_REGISTRATION_RECORD{
        PEXCEPTION_REGISTRATION_RECORD Next;
        PEXCEPTION_DISPOSITION Handler;
}EXCEPTION_REGISTRATION_RECORD,*PEXCEPTION_REGISTRATION_RECORD;
```

(2) SEH结构化异常处理 — SEH链

– 下面是一个异常处理函数的原型

➢异常处理函数有四个参数，这四个参数用来传递与异常相关的信息，包括异常类型、发生异常的代码地址、异常发生时CPU寄存器的状态等。

```
EXCEPTION_DISPOSITION _except_handler{
        EXCEPTION_RECORD *pRecord,
        EXCEPTION_REGISTRATION_RECORD *pFrame,
        CONTEXT *pContext,
        PVOID pValue
};
```

(2) SEH结构化异常处理 — SEH链

– 异常处理函数返回一个名为EXCEPTION_DISPOSITION的枚举类型，用于告知系统异常处理完成后程序应如何继续运行。

```
Typedef enum _EXCEPTION_DISPOSITION{
        ExceptionContinueExecution =0,//继续执行异常代码
        ExceptionContinueSearch =1,//运行下一个异常处理器
        ExceptionNestedException =2,//在OS内部使用
        ExceptionCollidedUnwind =3//在OS内部使用
}EXCEPTION_DISPOSITION;
```

(2) SEH结构化异常处理 — SEH的应用原理

2. SEH的应用原理

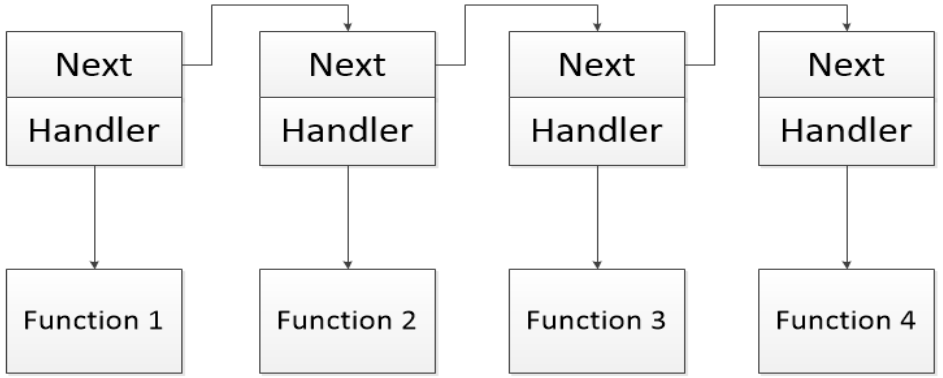
- 系统在使用SEH链时需要知道SEH链的地址。
- SEH链的头部地址储存在TEB中。
  - TEB是线程描述块，存储着线程运行所需的各种信息，例如线程的空间大小、寄存器状态、堆栈地址等。
  - 在TEB的第一个DWORD成员中存储着SEH链表头的地址，系统就可以通过这个地址找到SEH链。

(2) SEH结构化异常处理 — SEH的应用原理

- 当进程发生异常时，系统首先找出发生异常的线程，并根据该线程TEB中的信息获得第一个异常处理器的地址。
- 之后，异常被提交给进程的SEH链的第一个异常处理器，由异常处理函数对异常进行处理。
- 如果第一个异常处理器不能处理相关异常，则异常会被传递到下一个异常处理器，直到异常得到处理或进程SEH链结束，将异常抛给操作系统。
- 如果异常得到处理则程序继续运行，如果异常抛给操作系统，则系统会终止程序运行。

(2) SEH结构化异常处理 — SEH的应用原理

- 系统首先通过TEB获取第一个异常处理器的地址，然后将程序异常交由第一个异常处理器处理
- 如果Function1无法处理异常，则将异常传递到第二个异常处理器，如果Function2也无法处理异常，则继续传递下去，直到某个异常处理器能解决该异常。
- 如果传递到第四个异常处理器，Function4仍无法处理异常，则该异常将被抛给操作系统，由操作系统默认的异常处理进行处理。



(2) SEH结构化异常处理 — SEH的应用原理

- TEB（Thread Environment Block，线程环境块）系统在此TEB中保存频繁使用的线程相关的数据。
- 进程中的每个线程都有自己的一个TEB。一个进程的所有TEB都以堆栈的方式，存放在从0x7FFDE000开始的线性内存中，每4KB为一个完整的TEB，不过该内存区域是向下扩展的。
- 在用户模式下，当前线程的TEB位于独立的4KB段，可通过CPU的FS寄存器来访问该段，一般存储在[FS:0]。

(2) SEH结构化异常处理 — 在程序中添加SEH

3. 在程序中添加SEH

– SEH的添加基本都有以下三个步骤：

- 1）编写异常处理函数；
- 2）构造异常处理器；
- 3）将异常处理器从表头添加到SEH链。

(2) SEH结构化异常处理 — 在程序中添加SEH

- C语言中，程序员只需要使用_try将要监视运行的代码包起来，在__except中编写异常处理代码，语法如下所示
- SEH添加的其余工作交由编译器完成

```
__try{
// guarded code
}
__except ( expression ){
// exception handler code
}
```

(2) SEH结构化异常处理 — 在程序中添加SEH

- 汇编语言中，则需要程序员自己按照步骤添加SEH
- 首先将异常处理函数地址压栈，然后将SEH链表表头地址压栈。此时ESP指向了新构建的异常处理器地址，因此将表头地址FS:[0]修改为ESP地址。

```
PUSH @Handler                ;将异常处理函数地址压栈: Handler
PUSH DWORD PTR FS:[0]        ;将SEH链表表头地址压到:Next
MOV DWORD PTR FS:[0],ESP     ;将表头地址改为新的表头地址
```

- 一. Windows异常处理机制
- 二. SEH结构化异常处理
- 三. SEH反调原理
- 四. 逆向分析

(3) SEH反调原理

- SEH反调原理
  - SEH反调的本质，就是利用程序在正常运行与调试运行的不同情况下，实现不同的异常处理操作。
    - 正常运行的进程发生异常时，在SEH机制作用下，OS会接收异常，然后调用进程中注册的SEH处理
    - 如果进程在调试运行中发生异常，调试器OD就会接收处理异常。

(3) SEH反调原理

- 编程人员将程序真正的逻辑，放到异常处理函数中，并在逻辑中添加一些混淆和跳转，以使调试变得艰难。
  - 正常运行的情况下
    - 该程序运行发生异常后，直接调用异常处理函数，实现真正的功能；
  - 在调试运行的情况下
    - 该程序运行发生异常后，异常被调试器OD等捕获
  - 调试者通过查看SEH链发现自定义的异常处理函数，这种方式很容易被发现。

(3) SEH反调原理

- 为避免调试状态被发现，可使用两个函数
  - 使用函数UnhandledExceptionFilter可以设置系统默认的异常处理器
    - 该函数只有在非调试状态下才会被调用
    - 将异常处理器设置为系统默认的异常处理器，使异常发生时直接使用默认异常处理器进行处理
  - 使用函数SetUnhandledExceptionFilter设置自定义的异常处理器
    - 该函数只有在非调试状态下才会被调用
    - 该函数将默认异常处理器设置为自定义的异常处理器。

第六章 异常处理

- 一. Windows异常处理机制
- 二. SEH结构化异常处理
- 三. SEH反调原理
- 四. 逆向分析

（4）逆向分析

- 实验一：使用c语言添加SEH
- 实验二：使用内联汇编块注册SEH
- 实验三：SEH链分析
- 实验四：使用UnhandledExceptionFilter实现反调

（4）逆向分析 — 实验一：使用c语言添加SEH

- 实验内容
  - 实验一为编程实验，要求在代码中设计异常，并使用c语言的__try、__except添加异常处理逻辑，使程序在出现异常时弹出对话框，并修正原有错误使程序继续运行。

（4）逆向分析 — 实验一：使用c语言添加SEH

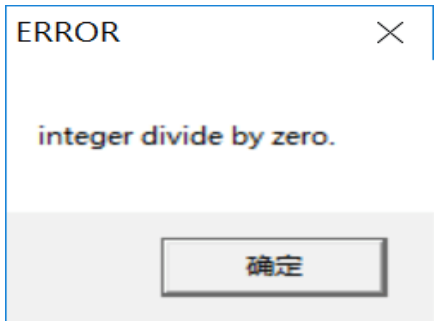
- 示例代码如下：

```
#include <stdio.h>
#include <windows.h>
int main(){
    int a =0, b =1, c =0;
    __try{
        c = b / a;//触发除零异常
    }
    __except(EXCEPTION_EXECUTE_HANDLER){
     MessageBox(NULL, TEXT("integer divide by zero."),TEXT("ERROR"), MB_OK);

            a =1;//修正的错误
            c = b / a;
     }
     printf("b / a = %d\n", c);
     system("pause");
     return 0;

}
```

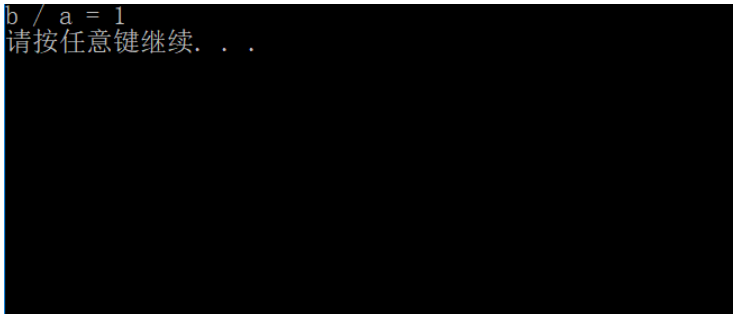
（4）逆向分析 — 实验一：使用c语言添加SEH

– 运行示例程序，程序触发除零异常，启动SEH，运行结果如下

➢启动SEH弹出对话框



➢修正错误后程序继续正常运行



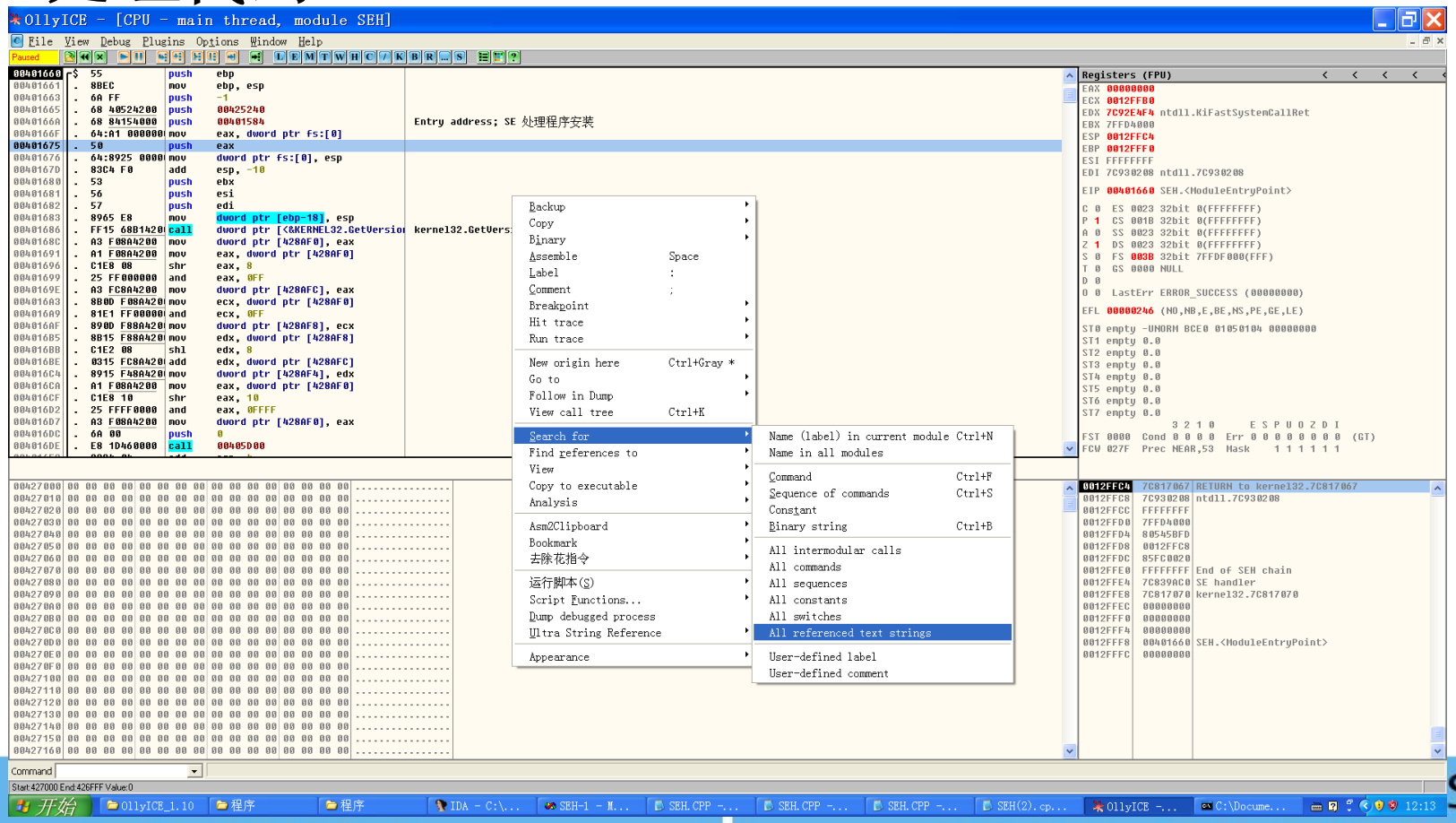
（4）逆向分析 — 实验一：使用c语言添加SEH

- 本实验示例展示了如何使用__try、__except添加SEH。
- 在使用关键字添加SEH时，由于编写者没有给出自定义的异常处理函数，编译器会生成一个特殊的函数，这个函数负责将_except中的代码注册为包装后的异常处理函数。
- 因为被包装了，因此在SEH中，使用__try、__except没有直接显示出添加的异常处理函数，而是显示为外面包着的那个特殊函数。

第六章 异常处理

(4) 逆向分析 — 实验一：使用c语言添加SEH

– 首先搜索字符串“integer divide by zero”找到异常处理代码



(4) 逆向分析 — 实验一：使用c语言添加SEH

– 双击选中的字符

OllyICE - [Text strings referenced in SEH:.text]

File View Debug Plugins Options Window Help

Paused

Address	Disassembly	Text string
00401038	push 00427A30	ASCII "SEH"
0040104E	push 00425024	ASCII "ok, you are here. Congratulation!",LF
0040105D	push 00425050	ASCII "ok, you are here. But your pw is wrong!",LF
0040106A	push 0042501C	ASCII "pause"
004010D6	push 004250B4	ASCII "please input password : "
004010E8	push 004250B0	ASCII "%s"
00401117	push 00425084	ASCII "What a pity, you found a wrong way.",LF
00401124	push 0042501C	ASCII "pause"
004011E5	push 004250F4	ASCII "format != NULL"
004011EE	push 004250E8	ASCII "printf.c"
004012EE	push 00425118	ASCII "The value of ESP was not properly saved across a function call.
004012FA	push 00425104	ASCII "i386\chkesp.c"
00401329	push 00425230	ASCII "COMSPEC"
00401376	push 00425218	ASCII "*command != _T('\0')"
0040137F	push 0042520C	ASCII "system.c"
0040139A	mov dword ptr [ebp-C], 00425208	ASCII "/c"
004013F8	mov dword ptr [ebp-1C], 004251FC	ASCII "command.com"

(4) 逆向分析 — 实验一：使用c语言添加SEH

– 右键选择搜索所有字符串，双击该字符串所在行则可以跳转到该字符串所在代码。

```
0040108A  PUSH OFFSET 00422050                    ASCII "ERROR"
0040108F  PUSH OFFSET 00422034                    ASCII "integer divide by zero."
004010BF  PUSH OFFSET 00422024                    ASCII "b / a = %d"
004010CC  PUSH OFFSET 0042201C                    ASCII "pause"
00401149  PUSH OFFSET 004220A4                    ASCII "COMSPEC"
00401196  PUSH OFFSET 0042208C                    ASCII "*command != _T('\0')"
0040119F  PUSH OFFSET 00422080                    ASCII "system.c"
004011BA  MOV DWORD PTR SS:[EBP-0C],OFFSET 004220 ASCII "/c"
00401218  MOV DWORD PTR SS:[EBP-1C],OFFSET 004220 ASCII "command.com"
00401221  MOV DWORD PTR SS:[EBP-1C],OFFSET 004220 ASCII "cmd.exe"
00401265  PUSH OFFSET 004220B8                    ASCII "format != NULL"
0040126E  PUSH OFFSET 004220AC                    ASCII "printf.c"
004013BC  ASCII "VC20XC00"
004014AE  PUSH OFFSET 004220DC                    ASCII "The value of ESP was not properly
```

(4) 逆向分析 — 实验一：使用c语言添加SEH

– 可见异常处理代码起始地址为**0x401083**

```
00401083  ·  8B65 E8       MOV ESP,DWORD PTR SS:[EBP-18]
00401086  ·  8BF4          MOV ESI,ESP
00401088  ·  6A 00         PUSH 0                                   Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
0040108A  ·  68 50204200   PUSH OFFSET 00422050                      Caption = "ERROR"
0040108F  ·  68 34204200   PUSH OFFSET 00422034                      Text = "integer divide by zero."
00401094  ·  6A 00         PUSH 0                                   hOwner = NULL
00401096  ·  FF15 D4724200 CALL DWORD PTR DS:[<&USER32.MessageBoxA  USER32.MessageBoxA
0040109C  ·  3BF4          CMP ESI,ESP
0040109E  ·  E8 FD030000   CALL 004014A0
004010A3  ·  C745 E4 0100  MOV DWORD PTR SS:[EBP-1C],1
004010AA  ·  8B45 E0       MOV EAX,DWORD PTR SS:[EBP-20]
004010AD  ·  99            CDQ
004010AE  ·  F77D E4       IDIV DWORD PTR SS:[EBP-1C]
004010B1  ·  8945 DC       MOV DWORD PTR SS:[EBP-24],EAX
004010B4  ·  C745 FC FFFF  MOV DWORD PTR SS:[EBP-4],-1
004010BB  >  8B4D DC       MOV ECX,DWORD PTR SS:[LOCAL.9]
004010BE  ·  51            PUSH ECX                                 Arg2 => [LOCAL.9]
004010BF  ·  68 24204200   PUSH OFFSET 00422024                     Arg1 = ASCII "b / a = %d□"
004010C4  ·  E8 87010000   CALL 00401250                            exceptionTest6-1.00401250
004010C9  ·  83C4 08       ADD ESP,8
004010CC  ·  68 1C204200   PUSH OFFSET 0042201C                     ASCII "pause"
004010D1  ·  E8 6A000000   CALL 00401140
004010D6  ·  83C4 04       ADD ESP,4
004010D9  ·  33C0          XOR EAX,EAX
004010DB  ·  8B4D F0       MOV ECX,DWORD PTR SS:[LOCAL.4]
004010DE  ·  64:890D 0000  MOV DWORD PTR FS:[0],ECX
004010E5  ·  5F            POP EDI
004010E6  ·  5E            POP ESI
004010E7  ·  5B            POP EBX
004010E8  ·  83C4 64       ADD ESP,64
004010EB  ·  3BEC          CMP EBP,ESP
004010ED  ·  E8 AE030000   CALL 004014A0
004010F2  ·  8BE5          MOV ESP,EBP
004010F4  ·  5D            POP EBP
004010F5  L·  C3           RETN
```

（4）逆向分析 — 实验一：使用c语言添加SEH

- 运行程序，在异常触发时查看程序的SEH链，如图
- 可见SEH链中并没有出现异常处理代码的起始地址（0x401083），因为异常处理代码的调用被包含在了地址0x4013C4的代码中。

指数	类型	链接	处理程序
1	SEH	0019FF30	004013C4
2	SEH	0019FF70	004013C4
3	SEH	0019FFCC	7732ED70
4	SEH	0019FFE4	7733B80A

(4) 逆向分析 — 实验二：使用内联汇编块注册SEH

- 实验内容
  - 实验二为编程实验，实现口令匹配程序。
  - 实验要求使用内联汇编块，注册自定义的异常处理函数，在异常处理函数中实现口令匹配功能
  - 因为一般调试人员不调试SEH链，这是利用SEH实现反调功能的一种方式。

（4）逆向分析 — 实验二：使用内联汇编块注册SEH

示例

– 异常处理函数

```
EXCEPTION_DISPOSITION
__cdecl
_except_handler(struct _EXCEPTION_RECORD *ExceptionRecord,
void* EstabliSEHrFrame,
struct _CONTEXT *ContextRecord,
void* DispatcherContext )
{
        //异常处理函数中口令匹配逻辑
        if(strcmp(input, pw)==0){
                printf("ok, you are here. Congratulation!\n");
        }
        else{
                printf("ok, you are here. But your pw is wrong!\n");
        }
        system("pause");
       exit(0);//完成口令匹配后结束进程—否则因为没有对异常代码进行处理，所以该进程会一直在触发异常和异常处理之间循环。

        //返回，告知os继续执行异常代码
        return ExceptionContinueExecution;
}
```

(4) 逆向分析 — 实验二：使用内联汇编块注册SEH

- 示例
  - 使用内联汇编块添加SEH

```
DWORD handler =(DWORD)_except_handler;
//注册异常处理函数
__asm{
        push handler;
        push FS:[0];
        mov FS:[0], ESP;
}
```

（4）逆向分析 — 实验二：使用内联汇编块注册SEH

– 运行示例程序，程序中会产生除零异常。产生异常后，程序会调用异常处理函数。异常处理函数根据口令匹配逻辑进行显示。输入错误口令，结果如下。

```
please input password : 123
ok, you are here. But your pw is wrong!
请按任意键继续. . .
```

(4) 逆向分析 — 实验二：使用内联汇编块注册SEH

– 输入正确口令，结果如下

```
please input password : SEH
ok, you are here. Congratulation!
请按任意键继续. . .
```

（4）逆向分析 — 实验二：使用内联汇编块注册SEH

- 实验二示例展示了如何利用SEH技术实现程序的反调功能。
- 在利用SEH实现反调功能时，程序真正的功能隐藏在异常处理函数中实现，并故意在主逻辑中触发异常来调用异常处理函数。

（4）逆向分析 — 实验二：使用内联汇编块注册SEH

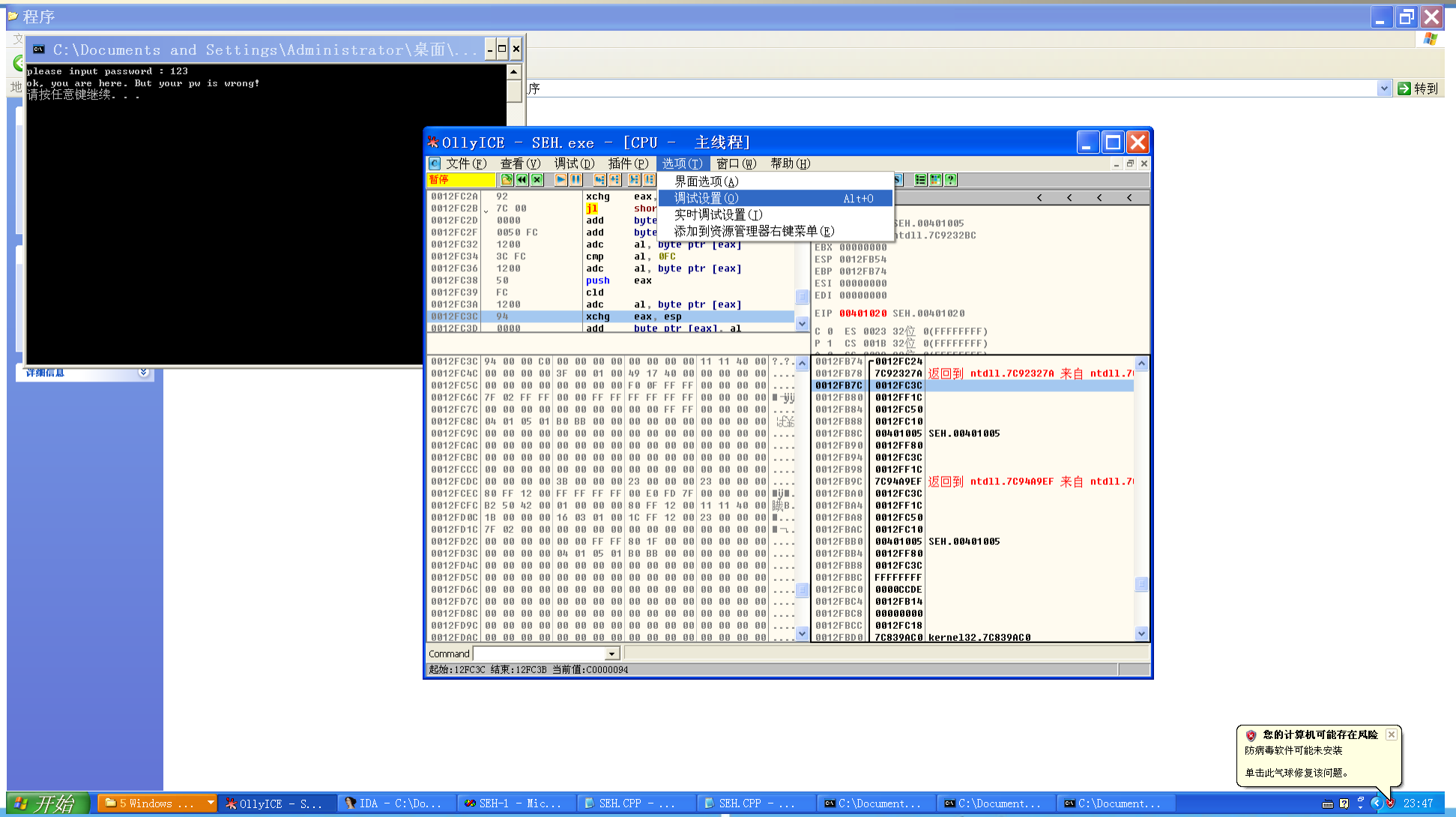
- 这是最简单的反调方式之一。一旦分析者对SEH进行分析，就暴露了。这个简单的隐藏在一般不调试的SEH链中。
- 通常情况下，为了反调，程序员一般不会采用内联汇编方式主动注册异常处理函数。
  - 通过内联汇编方式主动注册的异常处理函数会直接显示在SEH链中，很容易被发现；
  - 而像使用_try、_except添加的异常处理代码不会直接显示在SEH链中，而是编译器编译后添加到SEH链中，隐藏性高，这样会给调试带来一定的难度，从而达到反调的效果。
- 实验二这样做一方面使读者熟悉汇编代码添加SEH的操作，另一方面也方便实验三的讲解。

(4) 逆向分析 — 实验三：SEH链分析

- 实验内容
  - 实验三是分析实验，要求通过分析实验二的示例程序获取正确口令。
  - 要求通过实验了解SEH分析的基本流程，熟悉SEH安装的汇编代码，熟悉使用OD进行SEH链表查询和异常处理函数参数查看。

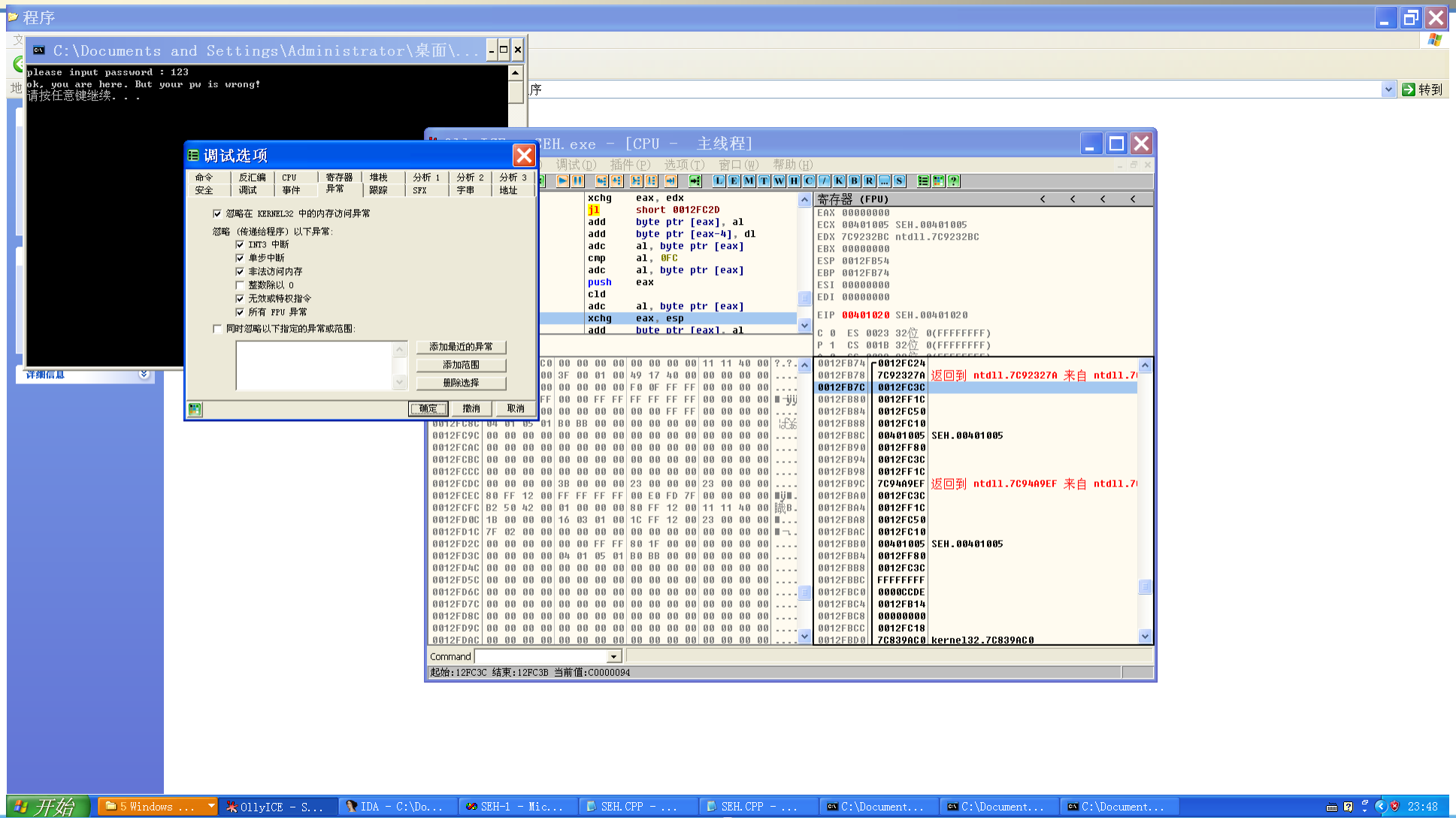
第六章 异常处理

(4) 逆向分析 — 实验三：SEH链分析



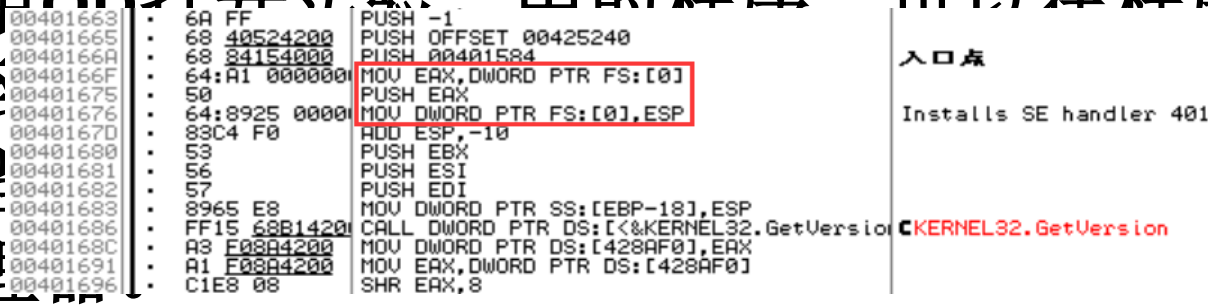
第六章 异常处理

(4) 逆向分析 — 实验三：SEH链分析

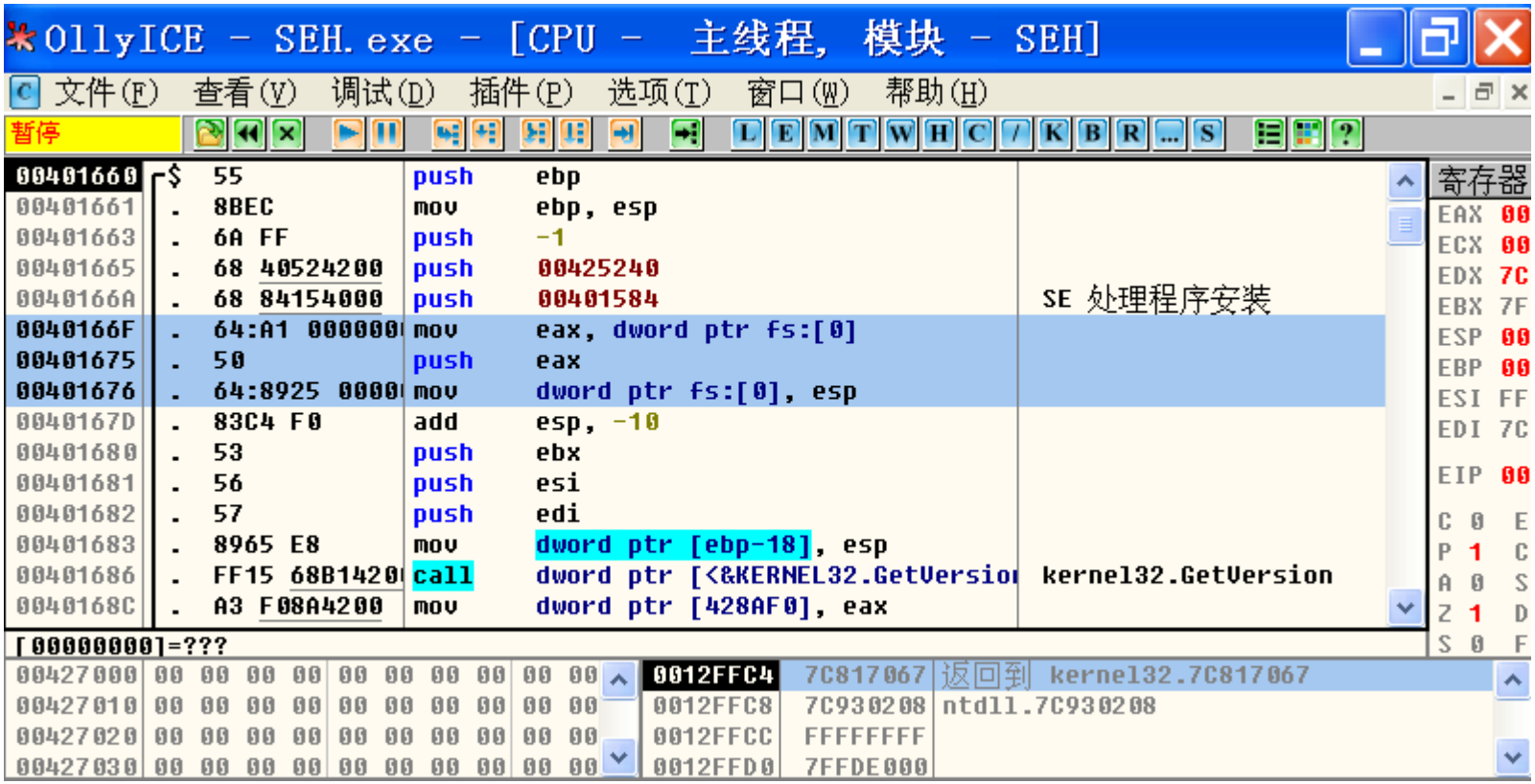


（4）逆向分析 — 实验三：SEH链分析

- 示例
  - 使用OD打开实验一中的程序，可以在程序入口位置看到不是示例程序的默认异常处理器
  - 其中，FS:[0]是SEH链表头的地址
  - SEH节点的添加是在链表头进行添加，因此会将原链表头地址赋给新节点的Next成员（即mov eax dword ptr FS:[0]），然后以新节点地址作为链表头地址(即mov dword ptr FS:[0] esp)。



（4）逆向分析 — 实验三：SEH链分析



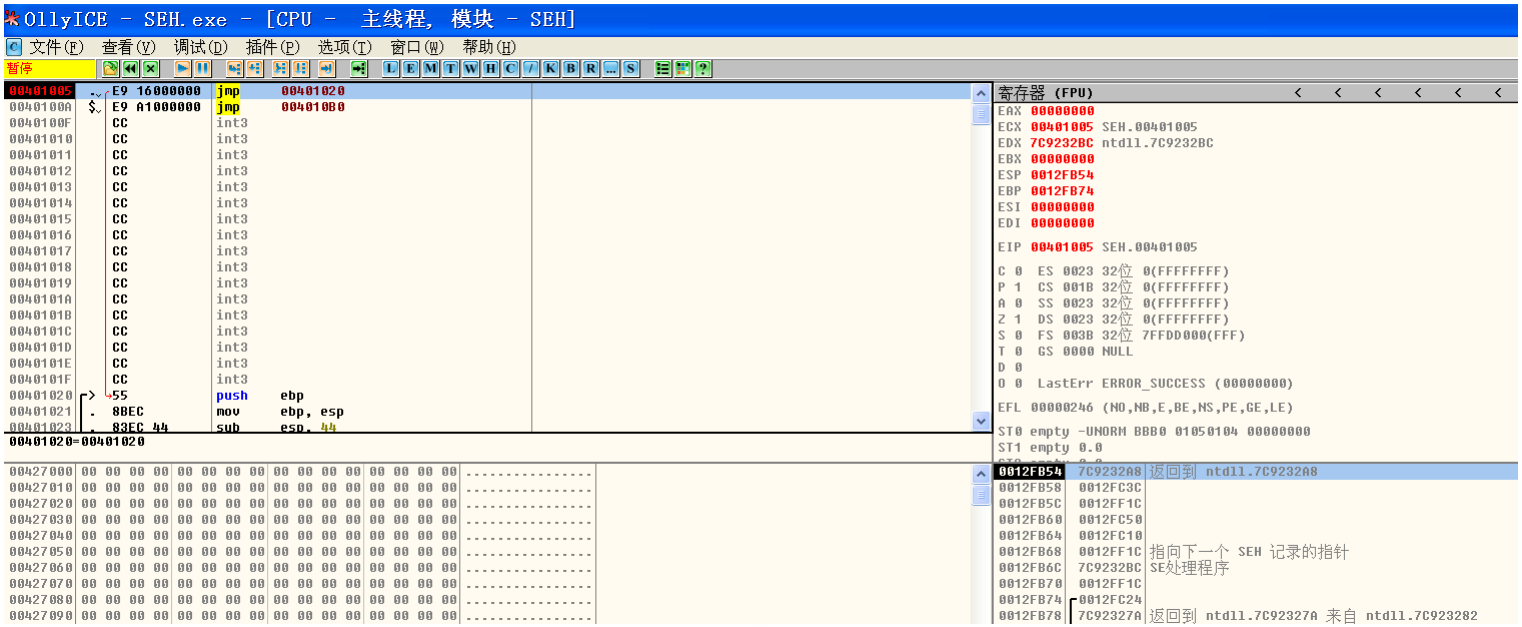
(4) 逆向分析 — 实验三：SEH链分析

- 将异常处理函数的地址和Next成员指向的地址压栈后（push eax），此时ESP的地址就是新节点的地址，因此第二条MOV指令将该地址作为新的表头地址赋给FS:[0] (即 mov dword ptr FS:[0] esp)
- 在软件分析过程中，如果看到对FS:[0]的操作，一般都与SEH有关。如果看到与FS:[0]有关的MOV指令，则很有可能在进行SEH的添加或删除。

```
00401663   .  6A FF        PUSH -1
00401665   .  68 40524200  PUSH OFFSET 00425240
0040166A   .  68 84154000  PUSH 00401584                    入口点
0040166F   .  64:A1 000000 MOV EAX,DWORD PTR FS:[0]
00401675   .  50           PUSH EAX
00401676   .  64:8925 0000 MOV DWORD PTR FS:[0],ESP         Installs SE handler 401584
0040167D   .  83C4 F0      ADD ESP,-10
00401680   .  53           PUSH EBX
00401681   .  56           PUSH ESI
00401682   .  57           PUSH EDI
00401683   .  8965 E8      MOV DWORD PTR SS:[EBP-18],ESP
00401686   .  FF15 68B1420 CALL DWORD PTR DS:[<&KERNEL32.GetVersio KERNEL32.GetVersion
0040168C   .  A3 F08A4200  MOV DWORD PTR DS:[428AF0],EAX
00401691   .  A1 F08A4200  MOV EAX,DWORD PTR DS:[428AF0]
00401696   .  C1E8 08      SHR EAX,8
```

(4) 逆向分析 — 实验三：SEH链分析

- 直接运行程序，程序提示输入密码，此时并不知道密码，所以随便输入内容。（示例程序没有进行缓冲区溢出处理，输入内容不能过多）
- 继续运行程序，程序触发除零异常。该异常被调试器捕获。



（4）逆向分析 — 实验三：SEH链分析

– 此时查看SEH链，可以直接找到异常函数地址设置断点。OD中可以使用“视图->VEH/SEH链”查看当前进程的SEH链。

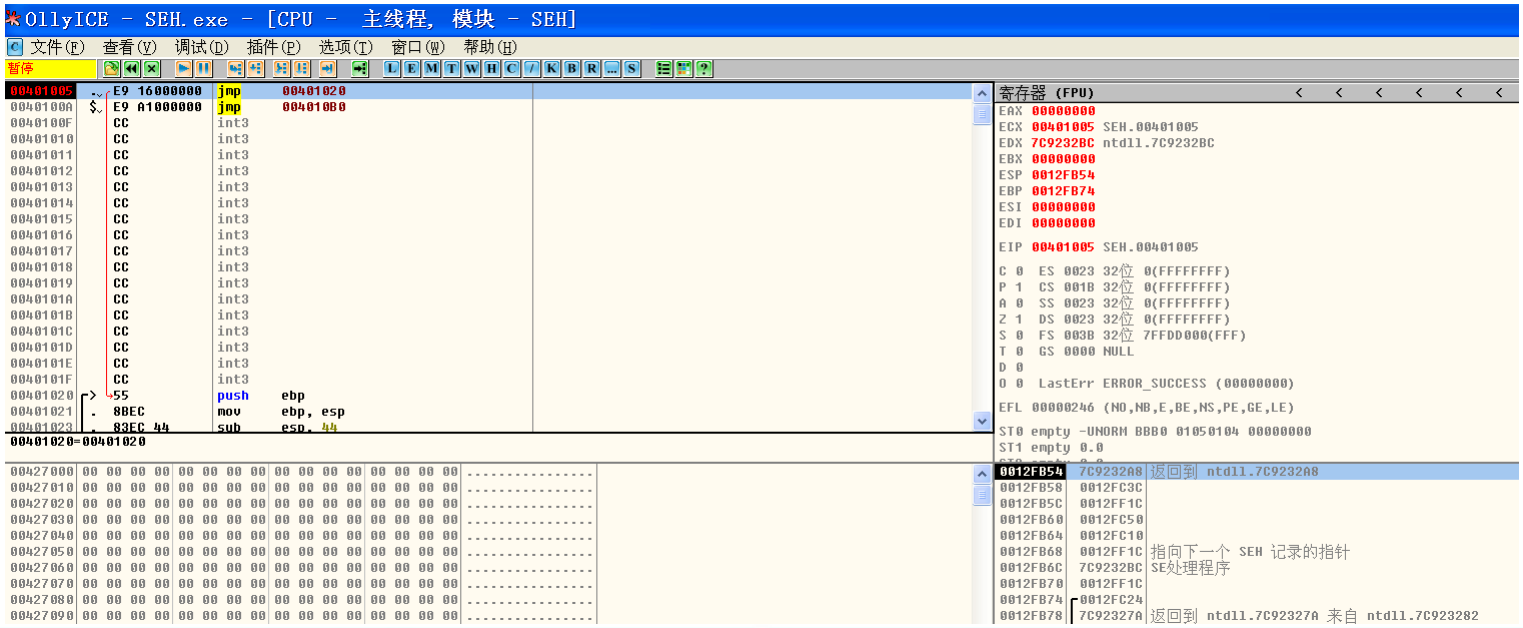
(4) 逆向分析 — 实验三：SEH链分析

- 因为SEH节点从链头开始添加，所以示例程序自定义的异常处理函数一定是第一个函数
- 最后一个是windows默认的异常处理函数。
- 在00401005第一个异常处理函数的地址处设置一个断点

1	SEH	0019FEDC	00401005
2	SEH	0019FF70	00401584
3	SEH	0019FFCC	77566A50
4	SEH	0019FFE4	77579ED9

(4) 逆向分析 — 实验三：SEH链分析

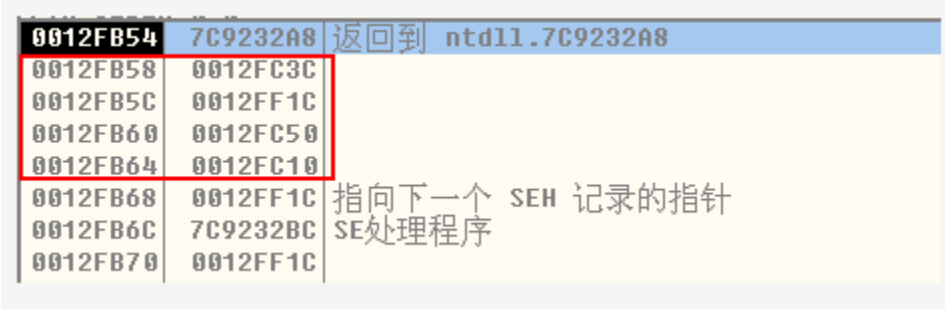
- 使用shift+F7/8/9将异常移交给被调程序，则被调程序会按照SEH链的顺序依次调用异常处理函数。
- 因为之前在异常处理函数处设了断点，所以程序会运行至异常处理函数起始地址。



（4）逆向分析 — 实验三：SEH链分析

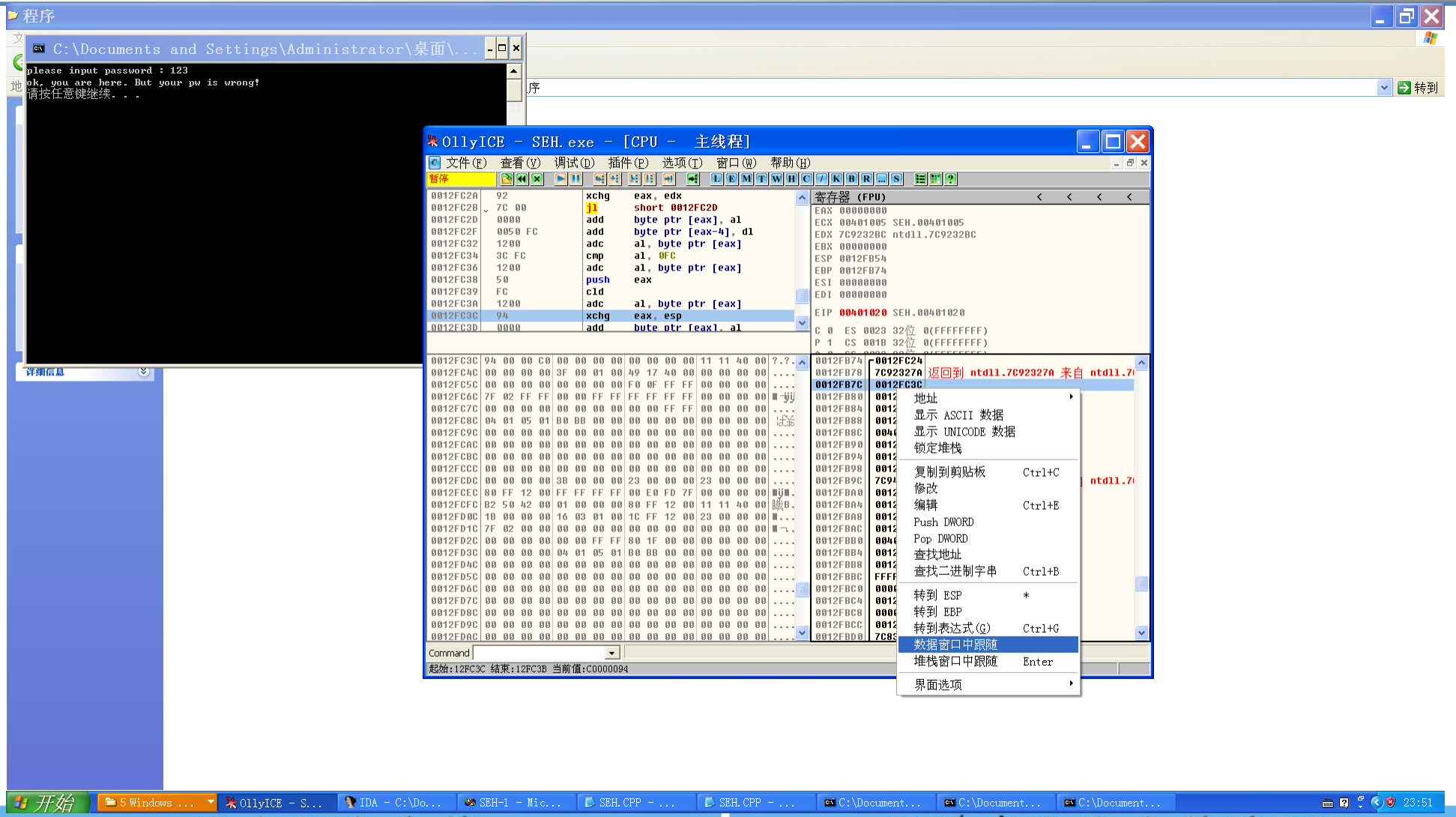
– 此时查看堆栈即可看到异常处理函数的参数。

➢(一般情况异常处理函数的参数分析意义不大，通常直接对函数代码进行分析，这里仅为了加深理解。)

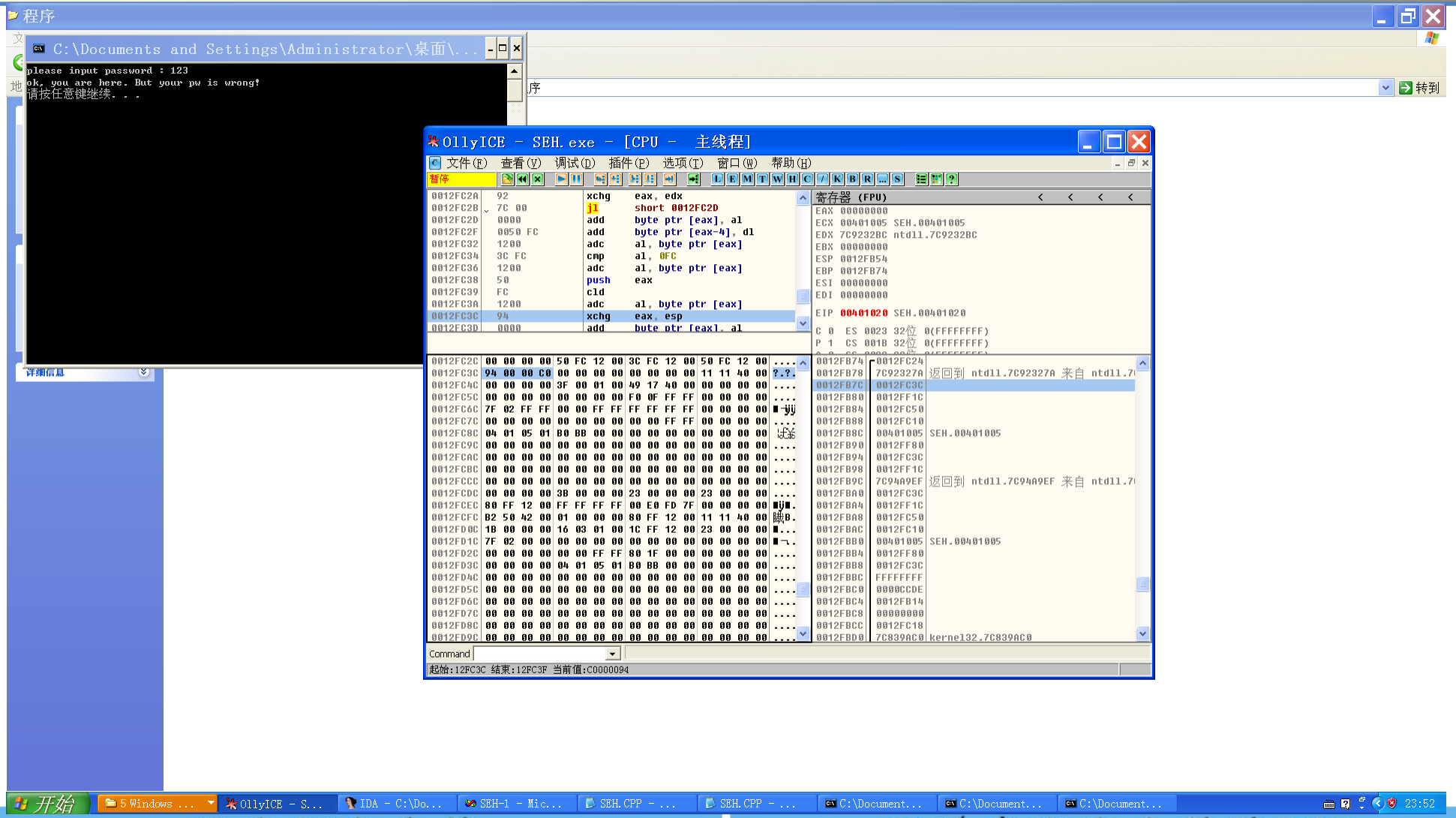


第六章 异常处理

(4) 逆向分析 — 实验三：SEH链分析



(4) 逆向分析 — 实验三：SEH链分析



（4）逆向分析 — 实验三：SEH链分析

– 下面是一个异常处理函数的原型

➢异常处理函数有四个参数，这四个参数用来传递与异常相关的信息，包括异常类型、发生异常的代码地址、异常发生时CPU寄存器的状态等。

```
EXCEPTION_DISPOSITION _except_handler{
        EXCEPTION_RECORD *pRecord,
        EXCEPTION_REGISTRATION_RECORD *pFrame,
        CONTEXT *pContext,
        PVOID pValue
};
```

（4）逆向分析 — 实验三：SEH链分析

- 第一个参数是指向EXPCEPTION_RECORD结构体的指针，根据指向的地址可以查看结构体中的数据
  - 第一个参数其中的第一个DWORD成员指出了异常类型代码为0xC0000094（即STATS_DIVIDE_BY_ZERO）。
- 第二个参数是指向EXCEPTION_REGISTRATION_RECORD结构体的指针，指向SEH链表。
- 第三个参数是指向CONTEXT结构体的指针，该结构体描述了异常发生时寄存器状态、异常发生的地址等上下文信息。
- 第四个参数仅供系统内部使用。

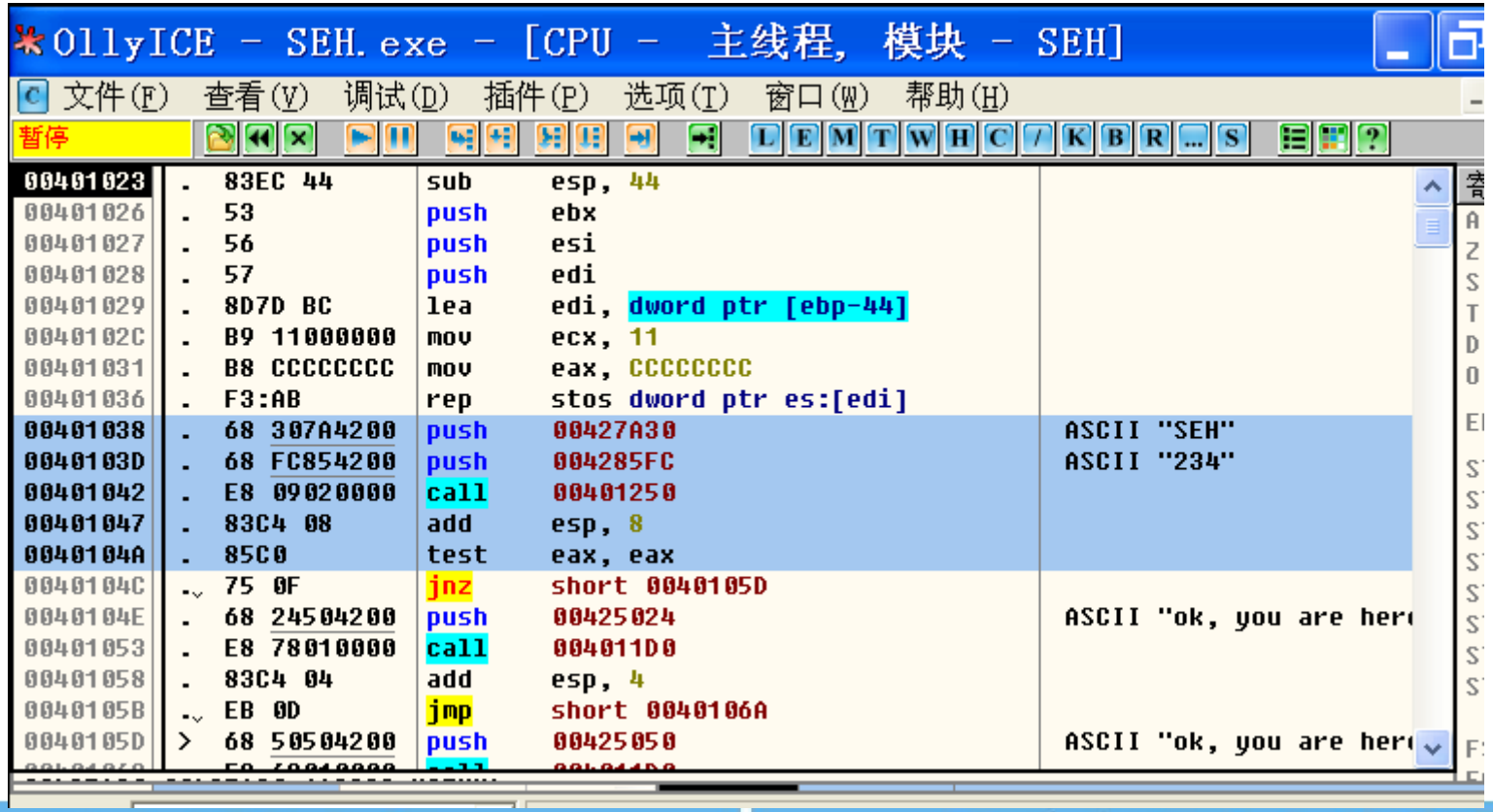
（4）逆向分析 — 实验三：SEH链分析

– 异常处理函数第一个参数是类型

0012FC2C	00	00	00	00	50	FC	12	00	3C	FC	12	00	50	FC	12	00	....P?.<?.P?.
0012FC3C	94	00	00	C0	00	00	00	00	00	00	00	00	11	11	40	00	?.?.....■■@.
0012FC4C	00	00	00	00	3F	00	01	00	49	17	40	00	00	00	00	00	....?.ふI■@.....
0012FC5C	00	00	00	00	00	00	00	00	F0	0F	FF	FF	00	00	00	00	.........?ÿÿ....
0012FC6C	7F	02	FF	FF	00	00	FF	FF	FF	FF	FF	FF	00	00	00	00	■ÿÿ..ÿÿÿÿÿÿ....
0012FC7C	00	00	00	00	00	00	00	00	00	00	FF	FF	00	00	00	00	..........ÿÿ....
0012FC8C	04	01	05	01	B0	BB	00	00	00	00	00	00	00	00	00	00	ばﾀ.....
0012FC9C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0012FCAC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0012FCBC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0012FCCC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0012FCDC	00	00	00	00	3B	00	00	00	23	00	00	00	23	00	00	00	....;...#...#...

(4) 逆向分析 — 实验三：SEH链分析

– 查看异常处理函数代码，可以直接看到口令比对代码。“123”是输入，所以“SEH”就是正确的口令了。



（4）逆向分析 — 实验三：SEH链分析

– 直接运行程序，输入“SEH”即可得到正确结果

```
please input password : SEH
ok, you are here. Congratulation!
请按任意键继续. . .
```

（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

- 实验内容
  - 实验四为编程实验，实现口令匹配程序。实验要求使用SetUnhandledExceptionFilter注册异常处理函数，在异常处理函数中实现口令匹配功能，也就是利用UnhandledExceptionFilter实现反调功能

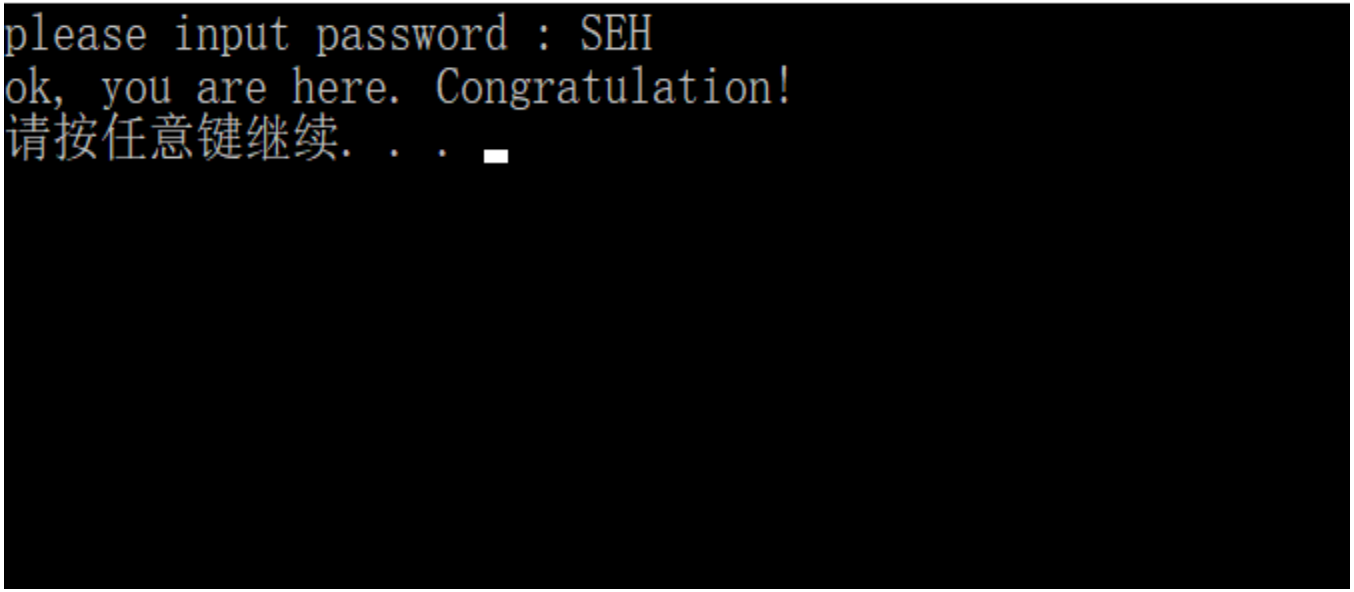
（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

- 示例
  - 示例程序运行效果与实验二效果相同。输入错误口令，结果如下。

```
please input password : 123
ok, you are here. But your pw is wrong!
请按任意键继续. . .
```

（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

– 输入正确口令，结果如下。



（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

– 使用OD对示例程序进行分析时,因为SetUnhandledExceptionFilter函数在调试状态下不起作用，所以不会调用自定义的异常处理函数，而且也不会在SEH链中看到自定义的异常处理函数。

（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

– 双击字符串所在行跳转到其所在代码，可见异常处理函数地址为0x401020

```
00401020  >  55            PUSH EBP
00401021  .  8BEC          MOV EBP,ESP
00401023  .  83EC 40       SUB ESP,40
00401026  .  53            PUSH EBX
00401027  .  56            PUSH ESI
00401028  .  57            PUSH EDI
00401029  .  8D7D C0       LEA EDI,[LOCAL.16]
0040102C  .  B9 10000000   MOV ECX,10
00401031  .  B8 CCCCCCCC   MOV EAX,CCCCCCCC
00401036  .  F3:AB         REP STOS DWORD PTR ES:[EDI]
00401038  .  68 307A4200   PUSH OFFSET 00427A30          ; Arg2 = ASCII "SEH"
0040103D  .  68 FC854200   PUSH OFFSET 004285FC          ; Arg1 = ASCII "123"
00401042  .  E8 99040000   CALL 004014E0                 ; SEH(2).004014E0
00401047  .  83C4 08       ADD ESP,8
0040104A  .  85C0          TEST EAX,EAX
0040104C  .v 75 0F         JNZ SHORT 0040105D
0040104E  .  68 58504200   PUSH OFFSET 00425058          ; ASCII "ok, you are here. Congratulation!"
00401053  .  E8 08040000   CALL 00401460
00401058  .  83C4 04       ADD ESP,4
0040105B  .v EB 0D         JMP SHORT 0040106A
0040105D  >  68 24504200   PUSH OFFSET 00425024          ; ASCII "ok, you are here. But your pw is wrong!"
00401062  .  E8 F9030000   CALL 00401460
00401067  .  83C4 04       ADD ESP,4
0040106A  >  68 1C504200   PUSH OFFSET 0042501C          ; Arg1 = ASCII "pause"
0040106F  .  E8 DC020000   CALL 00401350                 ; SEH(2).00401350
00401074  .  83C4 04       ADD ESP,4
00401077  .  6A 00         PUSH 0
00401079  .  E8 42010000   CALL 004011C0
0040107E  .  5F            POP EDI
0040107F  .  5E            POP ESI
00401080  .  5B            POP EBX
00401081  .  83C4 40       ADD ESP,40
00401084  .  3BEC          CMP EBP,ESP
00401086  .  E8 E5040000   CALL 00401570                 ; SEH(2).00401570
0040108B  .  8BE5          MOV ESP,EBP
0040108D  .  5D            POP EBP
0040108E  L  C3            RETN
```

（4）逆向分析 — 实验四：使用UnhandledExceptionFilter实现反调

- 运行程序，在异常触发时查看SEH链，如图。
- 可见0x401020地址不在SEH链中。
- 继续运行程序，按shift+F9/F8/F7，调试器会提示应用程序无法处理异常，因为调试状态下不会调用自定义的异常处理函数。

指数	类型	链接	处理程式
1	SEH	0019FF70	00407B54
2	SEH	0019FFCC	7732ED70
3	SEH	0019FFE4	7733B7E9

谢　谢!