通过 MAP 和 COD 文件找出程序崩溃位置

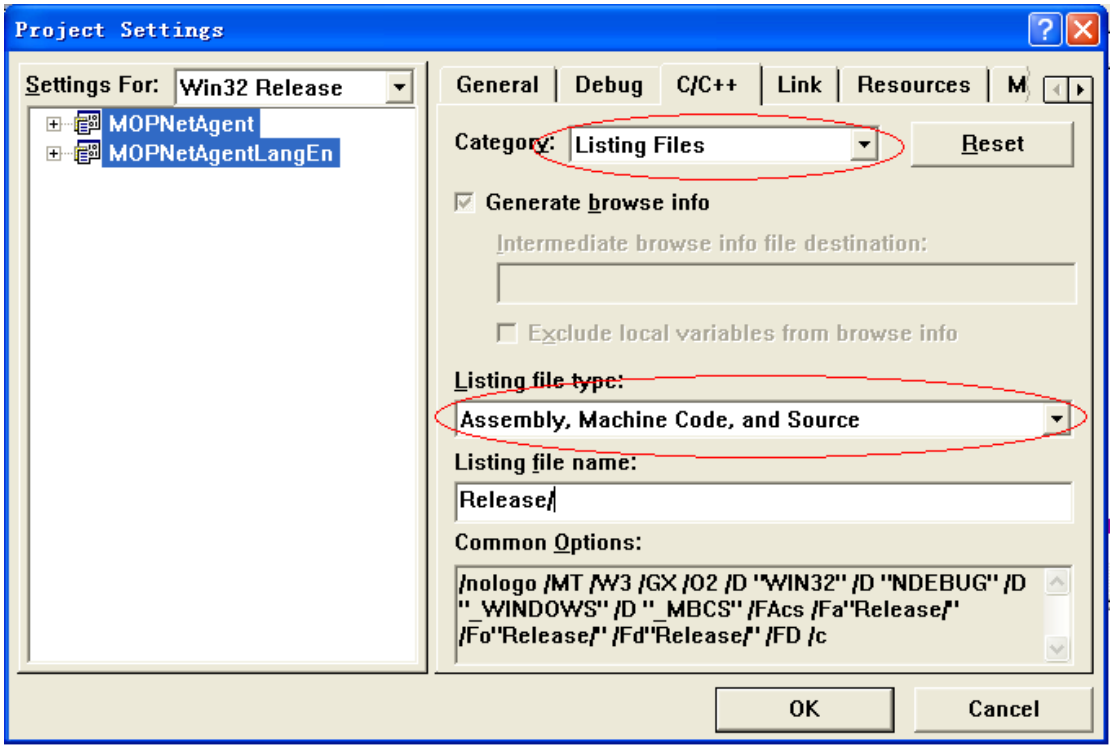
作者：刘智勇
liuzy57@sina.com

一、 编译器设置

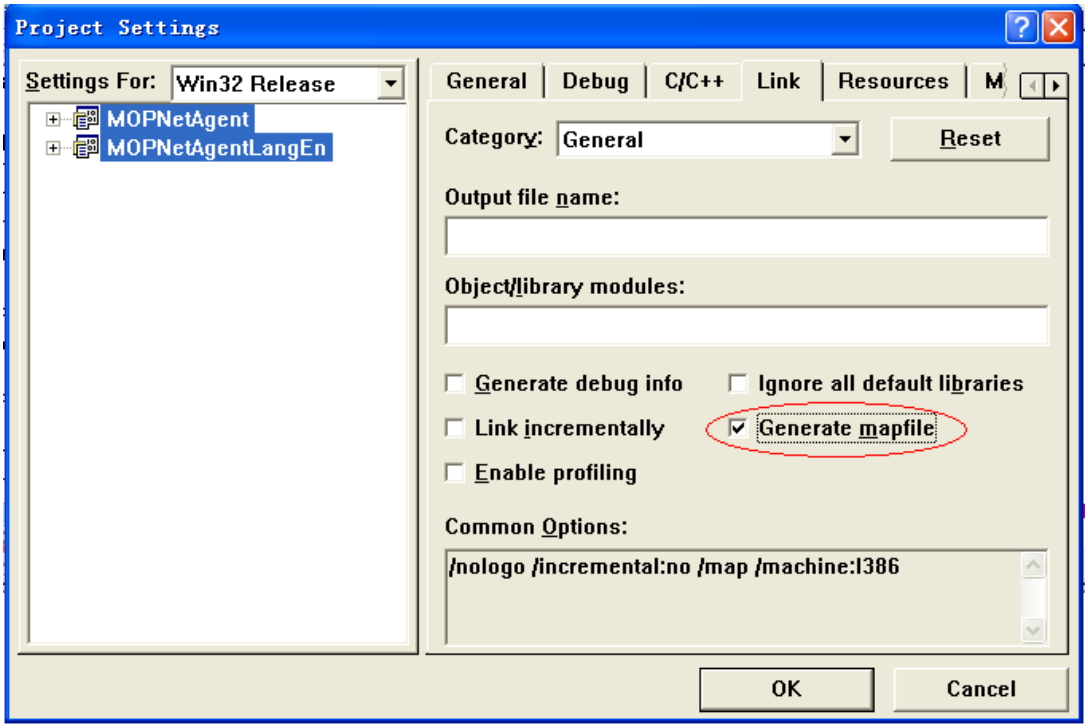
VC6:

选择菜单“**Project**”→”**Settings…**”，在弹出的”**Project Settings**”对话框中：

1. 选择”**C/C++**”属性页，然后在"**Category**"中选则"**Listing Files**"，再在"**Listing file type**"的组合框中选择"**Assembly, Machine code, and source**"。如下图:



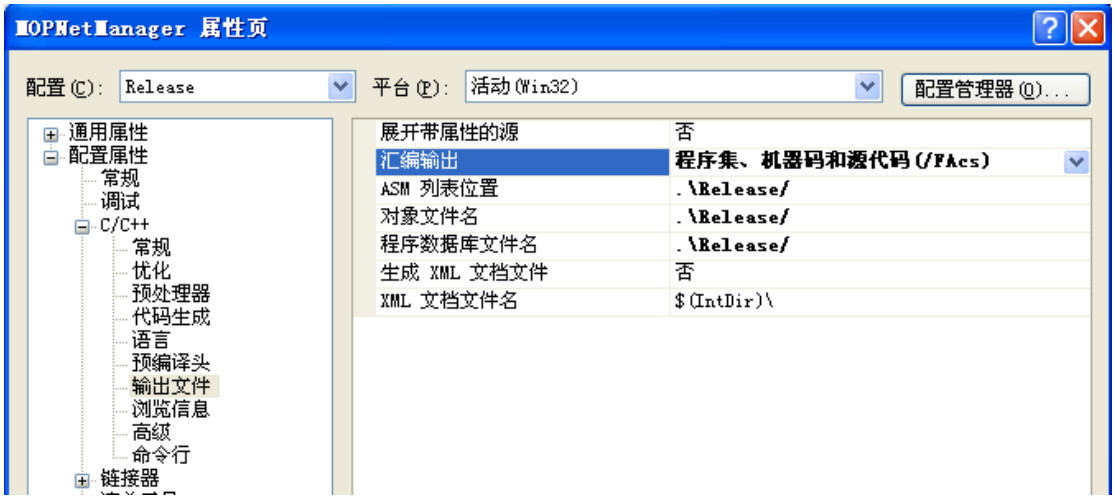
2. 选择”**Link**”属性页，在"**Category**"中选则"**General**"，然后选中”**Generate mapfile**”。如下图：



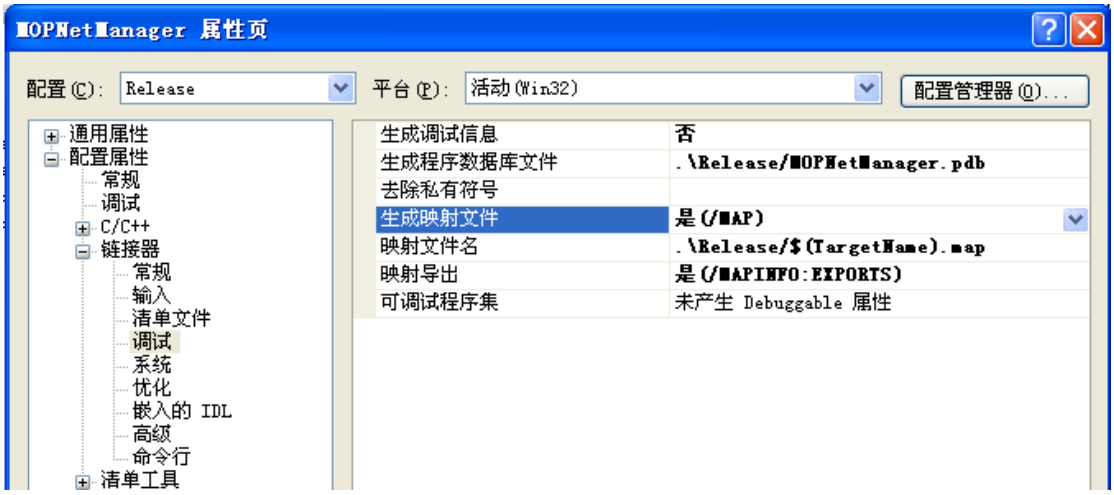
VC2005

选择菜单“**项目**”→”**属性**”，在弹出的属性页对话框中：

1. 选择”**配置属性**”→”**C/C++**”→”**输出文件**”，然后将”**汇编输出**”项选择为”**程序集、机器码和源代码/FAcs**”。如下图：



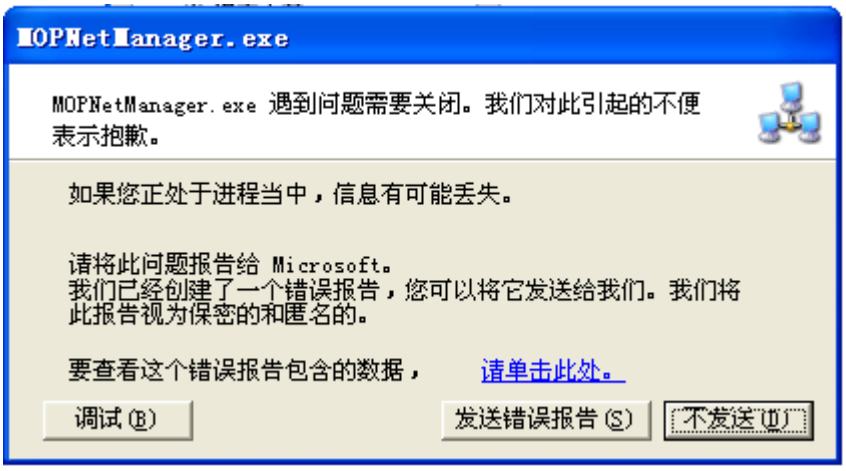
2. 选择”**配置属性**”→”**链接器**”→”**调试**”，然后将”**生成映射文件**”项选择为”**是/MAP**”。如下图：



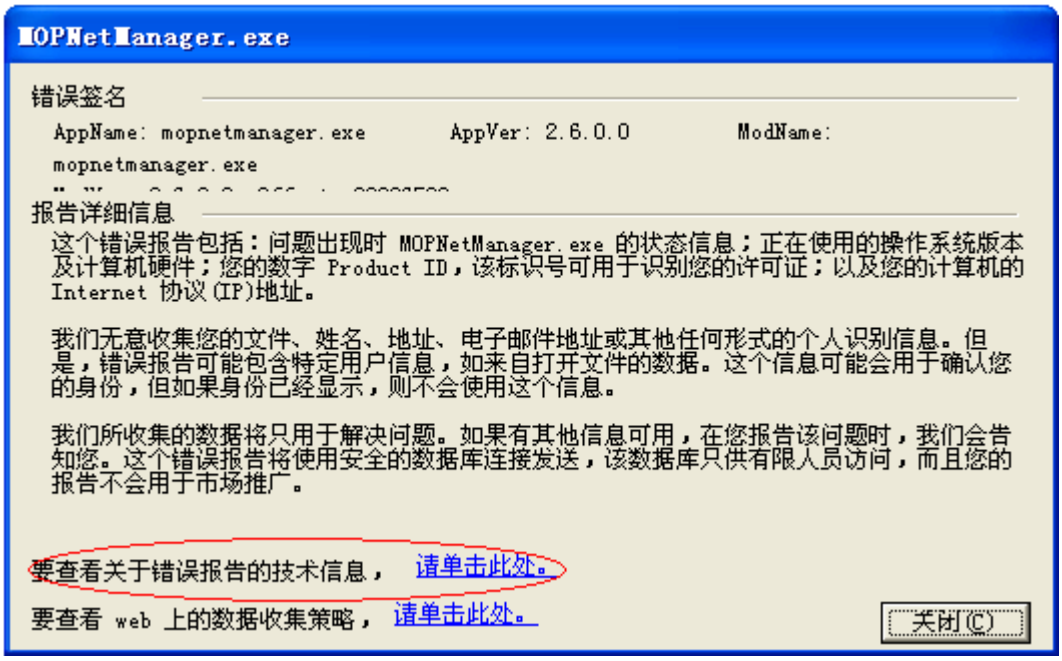
二、 重新编译整个项目

三、 定位崩溃位置(WinXP 系统，VC2005 项目)

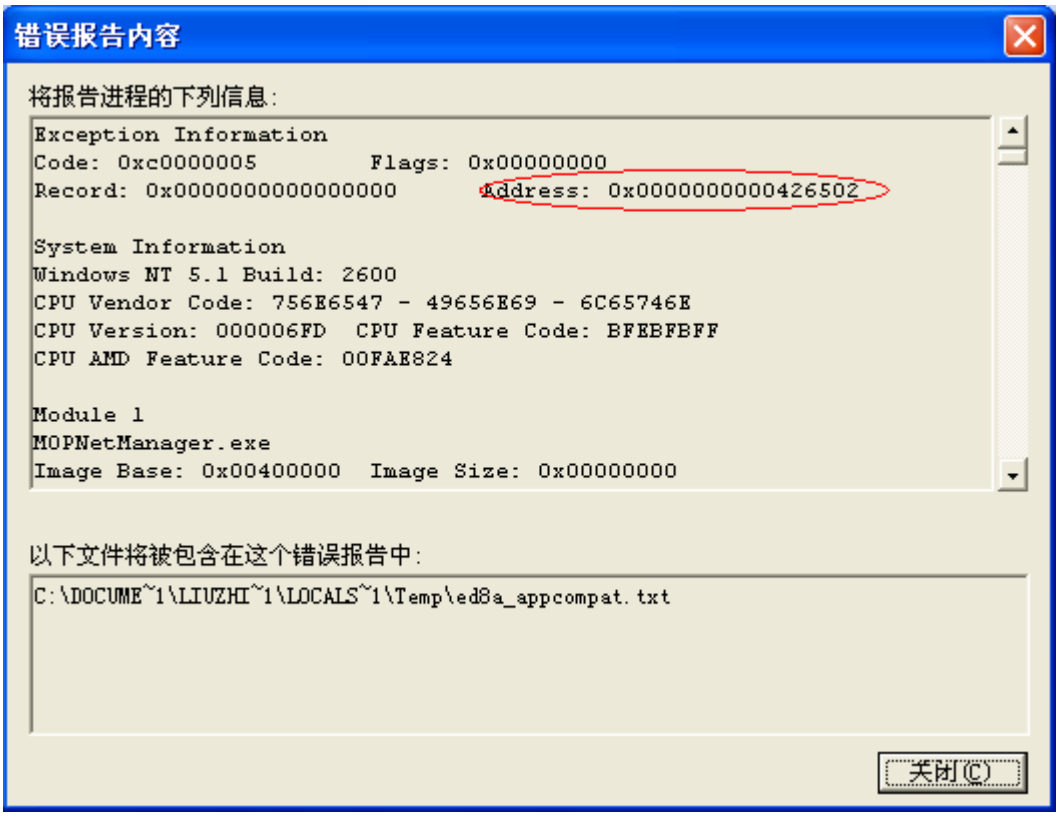
1. 运行程序过程中弹出程序崩溃对话框：



2. 点击”请单击此处。”，弹出如下提示框：



3. 选择”**要查看关于错误报告的技术信息**”对应的”请单击此处。”，弹出如下错误报告：



4. 记住程序的崩溃地址: 0x426502
5. 找开该程序对应的 MAP 文件，在”**Rva+Base**”项下查找小于此地址且最接近此地址的一个地址，如下图，可以看到是 004264c0：

```
0004:000023a0 00118e50H .rsrc$02                 DATA

 Address         Publics by Value              Rva+Base       Lib:Object
                                    ......
                                    ......
0001:000254a0       ?handle_input@CRemoteCtrlStreamer@@UAEHPAX@Z 004264a0 f   RemoteCtrlStreamer.obj
0001:000254c0       ?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z 004264c0 f   RemoteCtrlStreamer.obj
0001:00025510       ?DispatchNetEventLoop@CRemoteCtrlStreamer@@KGIPAX@Z 00426510 f   RemoteCtrlStreamer.obj
0001:00025530       ??_GCRemoteCtrlStreamer@@UAEPAXI@Z 00426530 f i RemoteCtrlStreamer.obj
0001:00025530       ??_ECRemoteCtrlStreamer@@UAEPAXI@Z 00426530 f i RemoteCtrlStreamer.obj
0001:00025550       ?new_stream@CRemoteCtrlStreamer@@SAPAV1@XZ 00426550 f   RemoteCtrlStreamer.obj
```

其对应的函数名为：**?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z**，由于所有以问号开头的函数名称都是 C++ 修饰的名称 ，"@....."则为区别重载函数而加的后缀，所以这个函数就是我们的源程序中 handle_close() 这个函数(也可看出是 CRemoteCtrlStreamer 类的成员函数)。也即是 handle_close()引起的崩溃，此函数的起始地址是 0x4264c0。

6. 定位出错行：打开该函数所在文件对应的 COD 文件，查找函数”?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z”的汇编代码，如下所示：

```
;   COMDAT ?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z
_TEXT   SEGMENT
_handle$ = 8                            ; size = 4
_close_mask$ = 12                       ; size = 4
?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z PROC  ;
CRemoteCtrlStreamer::handle_close, COMDAT
; _this$ = ecx

; 140  :    // TODO: Add your specialized code here.
; 141  :    //this->reactor()->remove_handler(this,     close_mask     |
ACE_Event_Handler::DONT_CALL);
; 142  :    TRACE(_T("in handle_close()\n"));
; 143  :    m_aceReactor.remove_handler(this,        close_mask        |
ACE_Event_Handler::DONT_CALL);

  00000 8b 44 24 08  mov     eax, DWORD PTR _close_mask$[esp-4]
  00004 56       push    esi
  00005 0d 00 02 00 00   or  eax, 512        ; 00000200H
  0000a 8b f1        mov     esi, ecx
  0000c 50       push    eax
  0000d 56       push    esi
  0000e b9 00 00 00 00   mov     ecx,
OFFSET      ?m_aceReactor@CRemoteCtrlStreamer@@0VACE_Reactor@@A       ;
CRemoteCtrlStreamer::m_aceReactor
  00013 ff 15 00 00 00
    00       call    DWORD                                          PTR
__imp_?remove_handler@ACE_Reactor@@UAEHPAVACE_Event_Handler@@K@Z

; 144  :    m_objStream.close();

  00019 8d 4e 20     lea     ecx, DWORD PTR [esi+32]
```

```
  0001c ff 15 00 00 00
    00       call    DWORD PTR __imp_?close@ACE_SOCK_Stream@@QAEHXZ

; 145  :    m_bConnected = FALSE;
; 146  :    /*notify upper that this link had been disconnected*/
; 147  :    if            (m_pfDisconnectedNotifyCbk!=NULL           &&
m_bConnectionStatusNotifyUpper)

  00022 8b 46 2c     mov     eax, DWORD PTR [esi+44]
  00025 85 c0        test    eax, eax
  00027 c7 46 38 00 00
    00 00        mov     DWORD PTR [esi+56], 0
  0002e 74 10        je  SHORT $LN1@handle_clo
  00030 83 7e 34 00  cmp     DWORD PTR [esi+52], 0
  00034 74 0a        je  SHORT $LN1@handle_clo

; 148  :    {
; 149  :        m_pfDisconnectedNotifyCbk(this,
m_pDisconnectedNotifyObj);

  00036 8b 4e 30     mov     ecx, DWORD PTR [esi+48]
  00039 51       push    ecx
  0003a 56       push    esi
  0003b ff d0        call    eax
  0003d 83 c4 08     add     esp, 8
$LN1@handle_clo:

; 150  :    }
; 151  :    delete this;

  00040 8b 16        mov     edx, DWORD PTR [esi]
  00042 8b 02        mov     eax, DWORD PTR [edx]
  00044 6a 01        push    1
  00046 8b ce        mov     ecx, esi
  00048 ff d0        call    eax

; 152  :    TRACE(_T("had deleted a stream!\n"));
; 153  :    return 0;

  0004a 33 c0        xor     eax, eax
  0004c 5e       pop     esi

; 154  : }
```

```
  0004d c2 08 00      ret     8
?handle_close@CRemoteCtrlStreamer@@UAEHPAXK@Z ENDP  ;
CRemoteCtrlStreamer::handle_close
_TEXT   ENDS
```

说明：

```
; 144   :    m_objStream.close();
```

冒号后表示源文件中的语句，冒号前的"144"表示该语句在源文件中的行数。这之后显示该语句汇编后的偏移地址，二进制码，汇编代码。如：

```
  00019 8d 4e 20      lea     ecx, DWORD PTR [esi+32]
  0001c ff 15 00 00 00
    00        call    DWORD PTR __imp_?close@ACE_SOCK_Stream@@QAEHXZ
```

其中"0019"表示相对于函数开始地址后的偏移，"8d 4e 20"为编译后的机器代码，" lea   ecx, DWORD PTR [esi+32]"为汇编代码。从"cod"文件中我们可以看出，一条(c/c++)语句通常需要编译成数条汇编语句 。此外有些汇编语句太长则会分两行显示如:

```
0001c   ff 15 00 00 00
    00       call     DWORD PTR __imp_?close@ACE_SOCK_Stream@@QAEHXZ
```

7. 计算崩溃地址相对于崩溃函数的偏移：

崩溃偏移地址=崩溃语句地址-崩溃函数的起始地址=0x426502-0x4264c0=0x42

8. 从 COD 文件中该函数的第一条语句的汇编指令：

```
  00000 8b 44 24 08   mov     eax, DWORD PTR _close_mask$[esp-4]
```

可看出函数开始的偏移地址等于 0x0000，因此，

崩溃语句的偏移=函数开始的偏移地址+崩溃偏移地址=0x0000+0x42=0x42

9. 查看相对偏移地址为 0x42 的代码：

```
  00042 8b 02         mov     eax, DWORD PTR [edx]
```

而该代码对应的 C++语句为：delete this;

据此可以确认是 delete this 语句导致的程序崩溃。

四、 参考文档：

[对“仅通过崩溃地址找出源代码的出错行”一文的补充与改进]