

Windows 调试工具入门-2

NetRoc

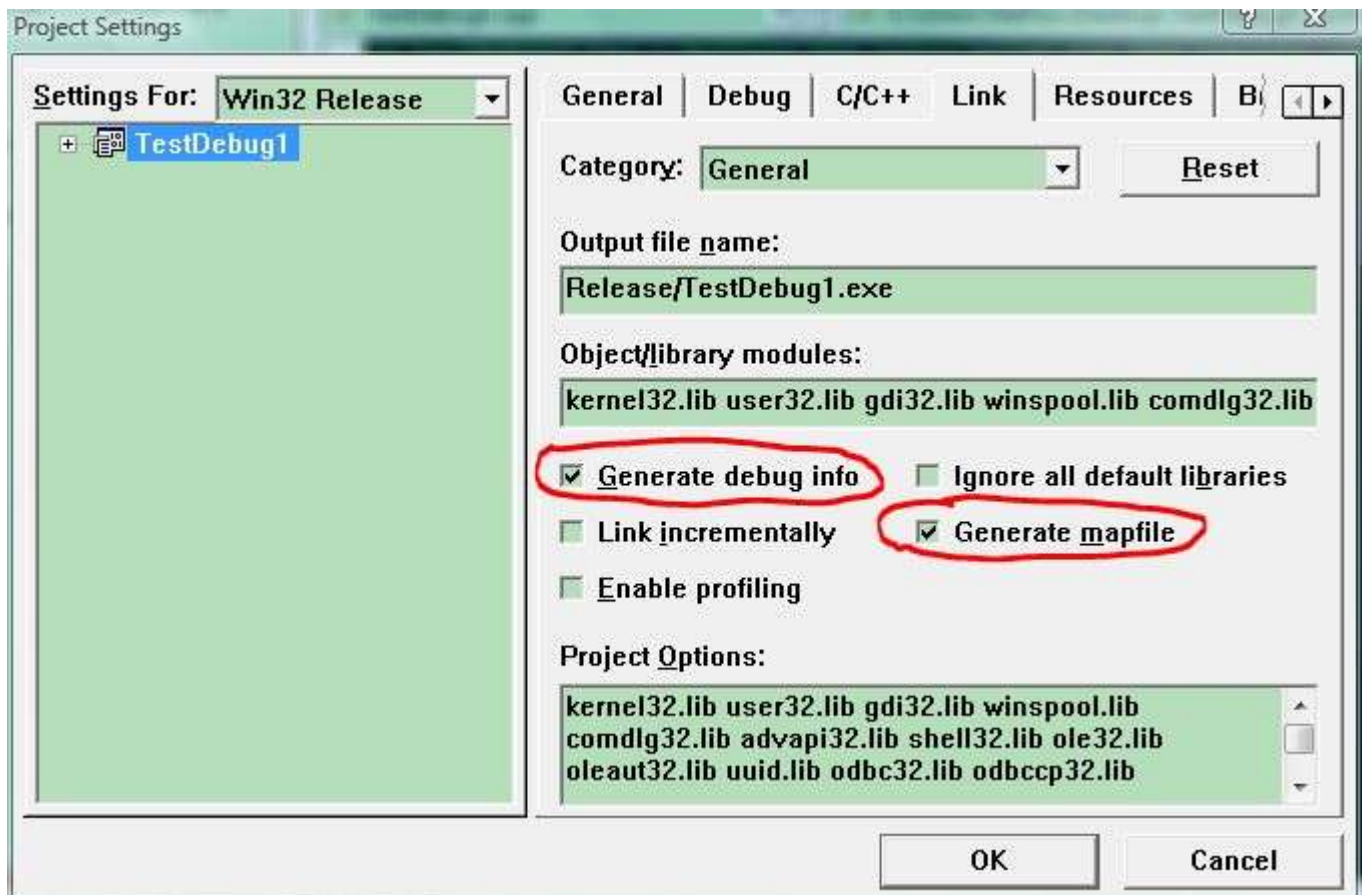
<http://www.DbgTech.net>

本篇介绍 Windows 调试工具的基本设置和基本操作方法。这里我们会用一个测试程序一步一步说明如何使用 WinDbg 开始调试工作。首先用 VC 建立一个名为 TestDebug1 的控制台项目，并生成它。

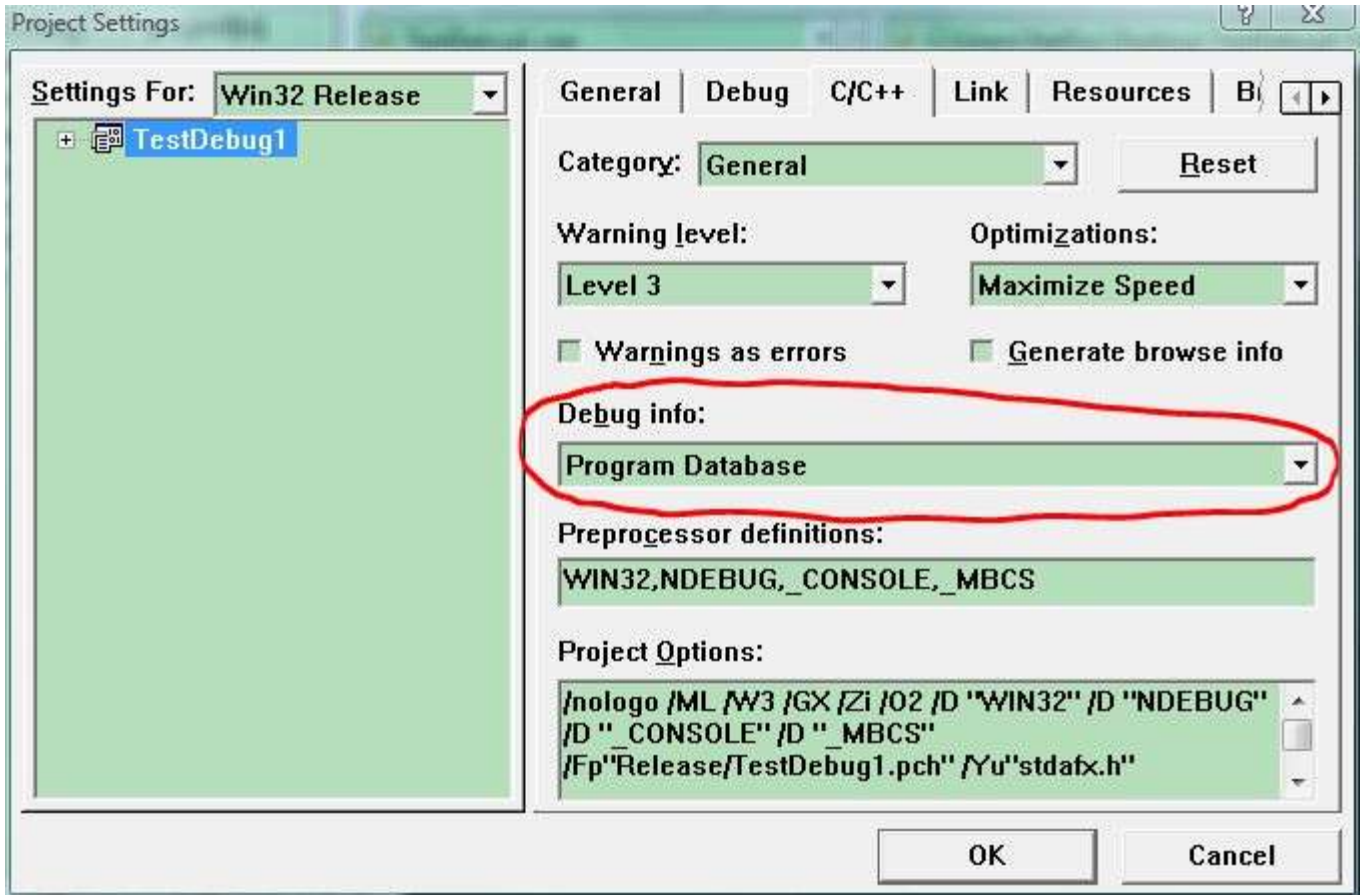
一、符号、源码和可执行映像路径设置

使用 WinDbg 开始调试工作之前，最重要的就是配置好各种环境了。这使得调试器可以正确识别调试目标中的各种变量、函数等等，使得我们能够进行符号化调试或者源码调试，而不是只能在一堆汇编代码中转圈。

首先来看一下未设置环境之前的样子。使用刚才说的 TestDebug1 项目，为了对比更清晰，用 Release 进行编译，链接选项中选中生成 map 文件和调试信息，如下：



在 C/C++选项卡中设置如下：



程序代码如下:

```
#include "stdafx.h"

#include <stdio.h>

int main(int argc, char* argv[])
{
    printf( "TestDebug1.cpp");

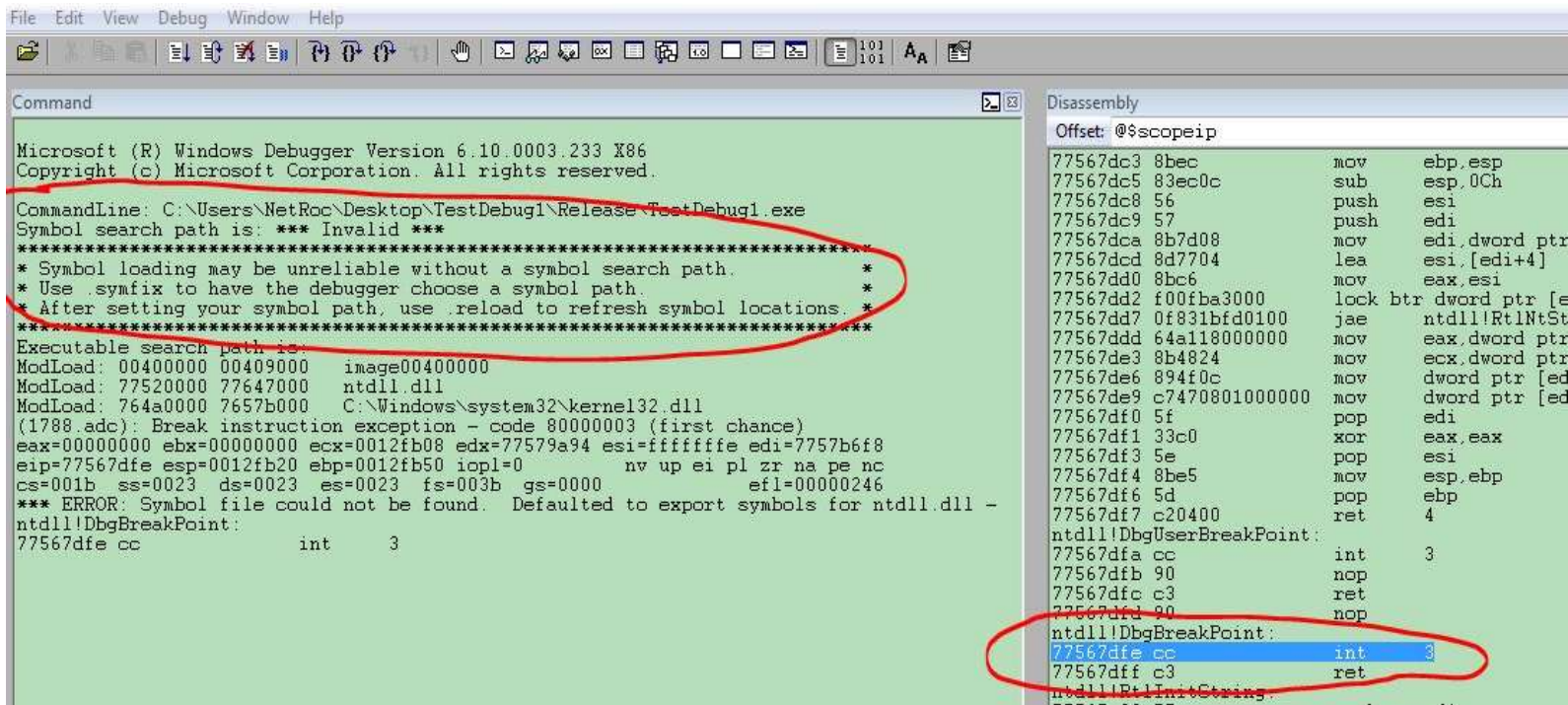
    return 0;
}
```

编译之后, 将 Release 目录下的 TestDebug1.pdb 剪切到其他目录下(如果没有这样做, 由于编译出来的程序中包含了符号文件路径, 调试器可以直接使用 exe 中的信息找到 pdb 文件, 而不需要设置路径)。在 map 文件中可以看到像下面这样的内容:

```
0001:00000000      _main                00401000 f    TestDebug1.obj
```

说明 main 函数位于 401000 地址处。

通过 WinDbg 的 File->Open Executable 菜单打开 TestDebug1.exe，可以在调试器命令窗口中看到下面的内容：



```
Microsoft (R) Windows Debugger Version 6.10.0003.233 X86
Copyright (c) Microsoft Corporation. All rights reserved.

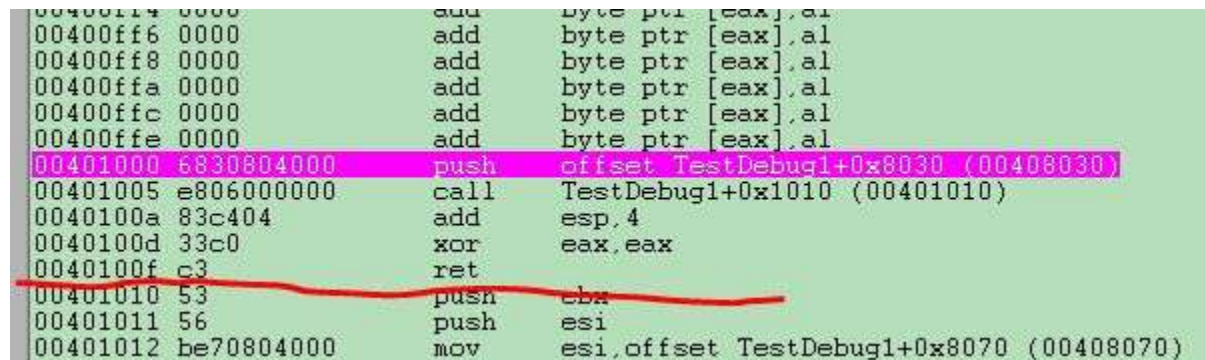
CommandLine: C:\Users\NetRoc\Desktop\TestDebug1\Release\TestDebug1.exe
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.
* Use .symfix to have the debugger choose a symbol path.
* After setting your symbol path, use .reload to refresh symbol locations.
*****
Executable search path is:
ModLoad: 00400000 00409000 image00400000
ModLoad: 77520000 77647000 ntdll.dll
ModLoad: 764a0000 7657b000 C:\Windows\system32\kernel32.dll
(1788.adc): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=0012fb08 edx=77579a94 esi=fffffffe edi=7757b6f8
eip=77567dfe esp=0012fb20 ebp=0012fb50 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
77567dfe cc          int     3

Disassembly
Offset: @$scopeip
77567dc3 8bec          mov     ebp,esp
77567dc5 83ec0c       sub     esp,0Ch
77567dc8 56          push    esi
77567dc9 57          push    edi
77567dca 8b7d08       mov     edi,dword ptr [edi+4]
77567dcd 8d7704       lea     esi,[edi+4]
77567dd0 8bc6       mov     eax,esi
77567dd2 f0fba3000   lock btr dword ptr [eax],0
77567dd7 0f831bfd0100 jae     ntdll!RtlNtSt
77567ddd 64a118000000 mov     eax,dword ptr [eax]
77567de3 8b4824       mov     ecx,dword ptr [eax]
77567de6 894f0c       mov     dword ptr [edi],ecx
77567de9 c7470801000000 mov     dword ptr [edi],eax
77567df0 5f          pop     edi
77567df1 33c0       xor     eax,eax
77567df3 5e          pop     esi
77567df4 8be5       mov     esp,ebp
77567df6 5d          pop     ebp
77567df7 c20400      ret     4
ntdll!DbgUserBreakPoint:
77567dfa cc          int     3
77567dfb 90          nop
77567dfc c3          ret
77567dfd 90          nop
ntdll!DbgBreakPoint:
77567dfe cc          int     3
77567dff c3          ret
ntdll!RtlInitString:
```

可以看到，调试器自动中断下来的位置并不是程序入口点，这是由 WinDbg 实现造成的，这里先不管它。

调试器命令窗口中可以看到，我们还没有设置符号路径，所以 WinDbg 目前还找不到 TestDebug1.exe 的任何符号文件。如果想在 main 函数下断，这时就不能使用符号，而只能直接使用 main 的地址。

使用命令 bp 00401000 在 main 函数设置断点，然后 F5 执行就可以中断到 main 的入口处了。断点设置和基本操作我们将在后面介绍。可以在反汇编窗口中看到这样的内容：



```
00400ff4 0000      add     byte ptr [eax],al
00400ff6 0000      add     byte ptr [eax],al
00400ff8 0000      add     byte ptr [eax],al
00400ffa 0000      add     byte ptr [eax],al
00400ffc 0000      add     byte ptr [eax],al
00400ffe 0000      add     byte ptr [eax],al
00401000 6830804000 push    offset TestDebug1+0x8030 (00408030)
00401005 e806000000 call    TestDebug1+0x1010 (00401010)
0040100a 83c404     add     esp,4
0040100d 33c0      xor     eax,eax
0040100f c3        ret
00401010 53        push    ebx
00401011 56        push    esi
00401012 be70804000 mov     esi,offset TestDebug1+0x8070 (00408070)
```

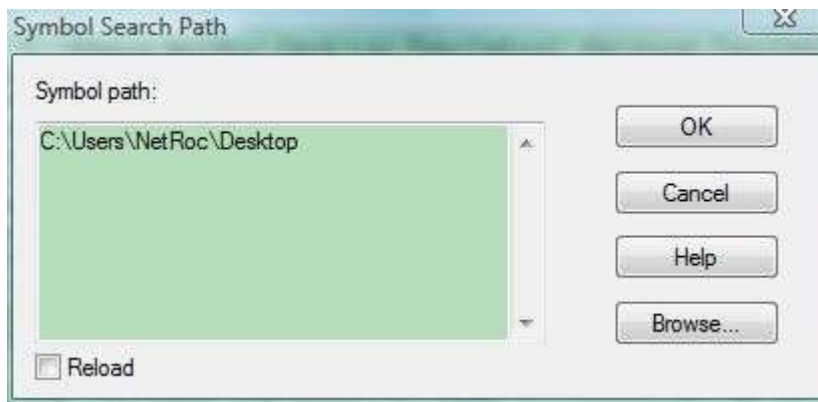
由于没有加载任何符号，所以我们看到的都是一堆反汇编代码和地址。在上一篇中已经介绍过，WinDbg 不像 OllyDbg 这些调试器一样拥有强大的反汇编分析能力，所以仅仅靠这些看起来一团乱麻的反汇编代码，调试工作是很难开展下去的。

- 符号路径的设置

要想在 WinDbg 中看到程序中的符号, 必须通过命令或者 WinDbg 菜单设置符号路径。如果还设置了 Microsoft 公共符号存储的话, 我们不但能够看到自己程序中的符号, 还能够看到 Windows 平台代码中的符号, 这对于调试会提供很好的帮助。

所谓符号路径, 就是包含了程序符号信息的符号文件所在的目录路径。通常我们接触到的符号文件都是以 `pdb` 作为后缀名的。`TestDebug1.exe` 项目如果在项目设置的 Link 选项中选中了生成调试信息的话(如上图中的 Generate debug info), 那么可以在 Debug 或者 Release 目录中找到它的符号文件 `TestDebug1.pdb`。

我们通过 WinDbg 的 File->Symbol File Path... 菜单, 或者命令 `.sympath` 设置符号路径为 `TestDebug1.pdb` 所在的目录。例如刚才我把生成的 `pdb` 文件移动到桌面上了, 所以在我的机器上就设置为:



完成之后在命令窗口输入 `.reload` 命令, 我们可以看到反汇编窗口的内容发生改变:

```
TestDebug1!main <PERF> (TestDebug1+0x11b):
00400ff6 0000      add     byte ptr [eax],al
TestDebug1!main <PERF> (TestDebug1+0xff8):
00400ff8 0000      add     byte ptr [eax],al
TestDebug1!main <PERF> (TestDebug1+0xffa):
00400ffa 0000      add     byte ptr [eax],al
TestDebug1!main <PERF> (TestDebug1+0xffc):
00400ffc 0000      add     byte ptr [eax],al
TestDebug1!main <PERF> (TestDebug1+0xffe):
00400ffe 0000      add     byte ptr [eax],al
TestDebug1!main:
00401000 6830804000 push    offset TestDebug1!`string' (00408030)
00401005 e806000000 call    TestDebug1!printf (00401010)
0040100a 83c404      add     esp,4
0040100d 33c0      xor     eax,eax
0040100f c3         ret
TestDebug1!printf:
00401010 53         push    ebx
00401011 56         push    esi
00401012 be70804000 mov     esi,offset TestDebug1!_iob+0x20 (00408070)
00401013 57         push    edi
```

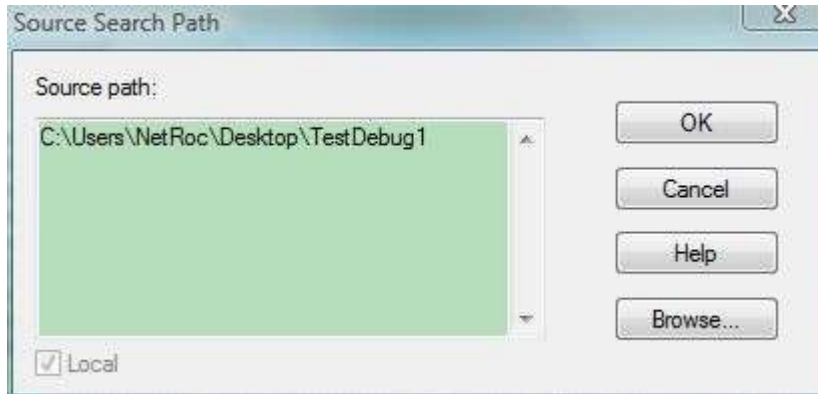
这里就已经可以看到 `TestDebug1.exe` 中的函数、变量名这样的符号了。而我们也可以通过 `bp main` 这样的命令直接使用符号来操作调试器。

另外, 在 Local、Watch 等窗口中也可以直接使用符号名查看到变量的值、在 Call Stack 窗口中可以看到函数名, 等等。

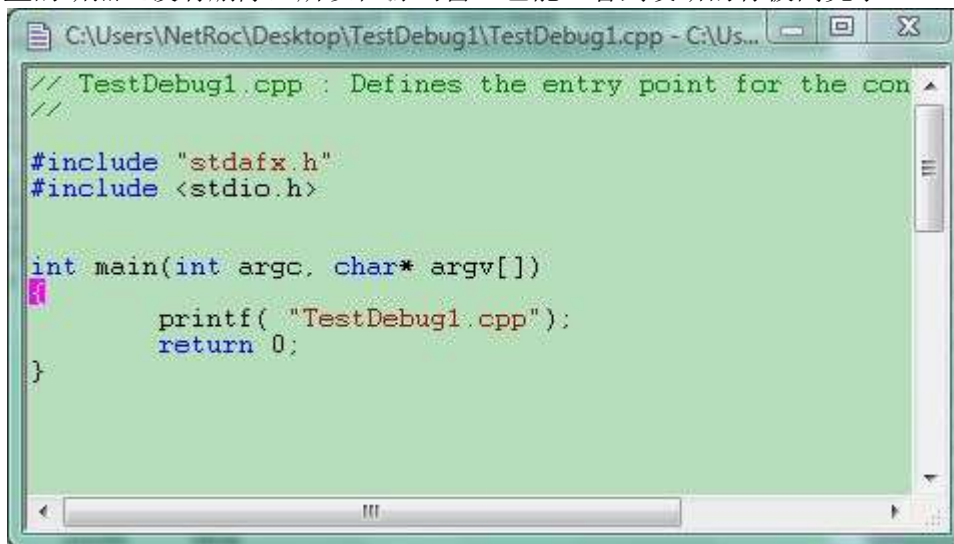
- 源码路径的设置

通过上面的设置，我们可以对程序进行符号化调试。如果拥有程序的代码，还可以通过设置源码路径来进行源码级调试。

继续上面的工作，我们通过 WinDbg 的 File->Source File Path...菜单或者.srspath 命令设置源代码保存的路径，比如我的机器上是这样：



确定之后，如果当前指令指针在源文件的代码范围内，就会自动跳出源文件窗口。如果没有跳出，那么可以通过 File->Open Source File...菜单手动打开源文件。由于刚才设置的断点还没有删除，所以在源码窗口也能看到设断的行被高亮了：



之后就基本上可以完全通过源码窗口进行设置断点、查看变量、跟踪代码等操作。比只有符号的时候方便了很多。

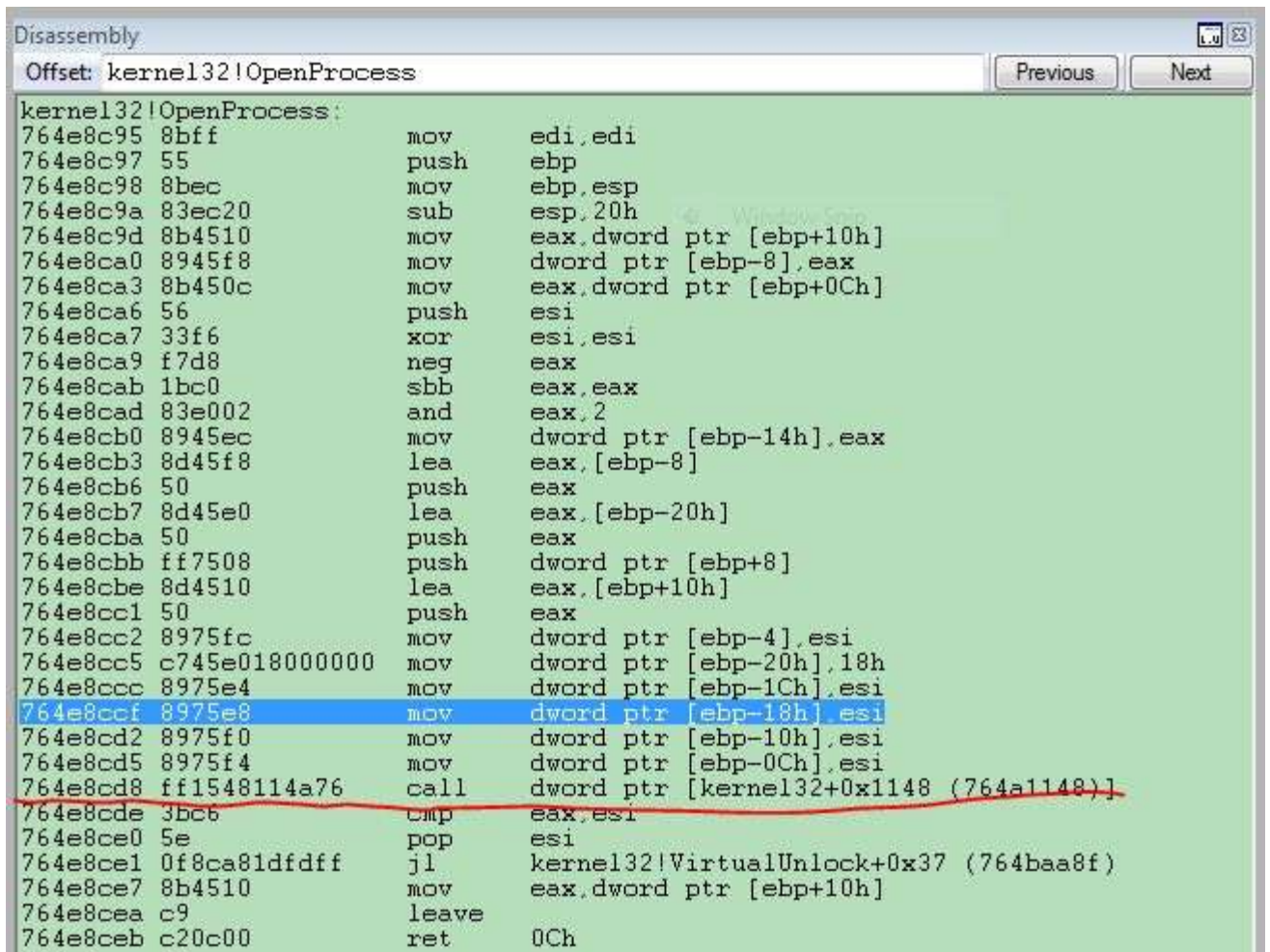
- 可执行映像路径的设置

可执行映像路径一般在调试 dump 文件时才用得上。需要将这个路径设置成要调试的 exe、dll、sys 等可执行文件的路径。可以通过 File->Image File Path...菜单或者.exepath 命令设置。

- 使用微软公共符号存储

除了使用自己程序的符号之外，调试时还可以使用微软提供的 Windows 系统代码的符号。这需要修改一下我们设置的符号路径。最方便的办法是使用.symfix 命令。

现在来看一下 kernel32.dll 中的代码，在反汇编窗口的 Offset 栏中填入 kernel32!OpenProcess，在我的机器上代码如下：

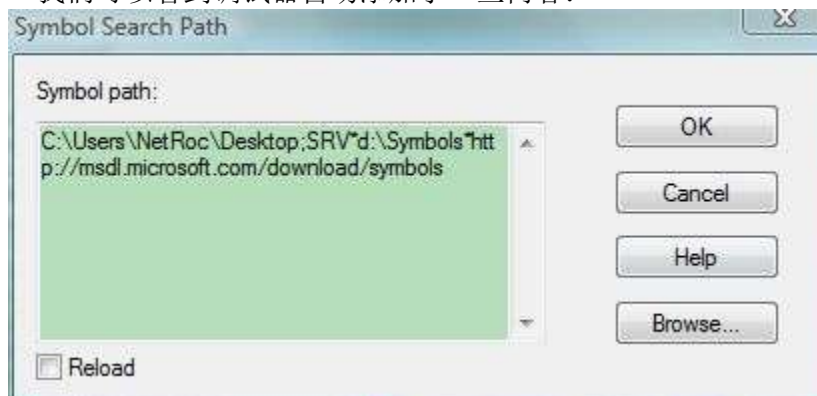


```
Disassembly
Offset: kernel32!OpenProcess
Previous Next

kernel32!OpenProcess:
764e8c95 8bff      mov     edi,edi
764e8c97 55        push   ebp
764e8c98 8bec      mov     ebp,esp
764e8c9a 83ec20    sub     esp,20h
764e8c9d 8b4510    mov     eax,dword ptr [ebp+10h]
764e8ca0 8945f8    mov     dword ptr [ebp-8],eax
764e8ca3 8b450c    mov     eax,dword ptr [ebp+0Ch]
764e8ca6 56        push   esi
764e8ca7 33f6      xor     esi,esi
764e8ca9 f7d8      neg     eax
764e8cab 1bc0      sbb     eax,eax
764e8cad 83e002    and     eax,2
764e8cb0 8945ec    mov     dword ptr [ebp-14h],eax
764e8cb3 8d45f8    lea     eax,[ebp-8]
764e8cb6 50        push   eax
764e8cb7 8d45e0    lea     eax,[ebp-20h]
764e8cba 50        push   eax
764e8cbb ff7508    push   dword ptr [ebp+8]
764e8cbe 8d4510    lea     eax,[ebp+10h]
764e8cc1 50        push   eax
764e8cc2 8975fc    mov     dword ptr [ebp-4],esi
764e8cc5 c745e018000000 mov     dword ptr [ebp-20h],18h
764e8ccc 8975e4    mov     dword ptr [ebp-1Ch],esi
764e8ccf 8975e8    mov     dword ptr [ebp-18h],esi
764e8cd2 8975f0    mov     dword ptr [ebp-10h],esi
764e8cd5 8975f4    mov     dword ptr [ebp-0Ch],esi
764e8cd8 ff1548114a76 call    dword ptr [kernel32+0x1148 (764a1148)]
764e8cde 3bc6      cmp     eax,esi
764e8ce0 5e        pop     esi
764e8ce1 0f8ca81dfdf jl      kernel32!VirtualUnlock+0x37 (764baa8f)
764e8ce7 8b4510    mov     eax,dword ptr [ebp+10h]
764e8cea c9        leave  eax
764e8ceb c20c00    ret     0Ch
```

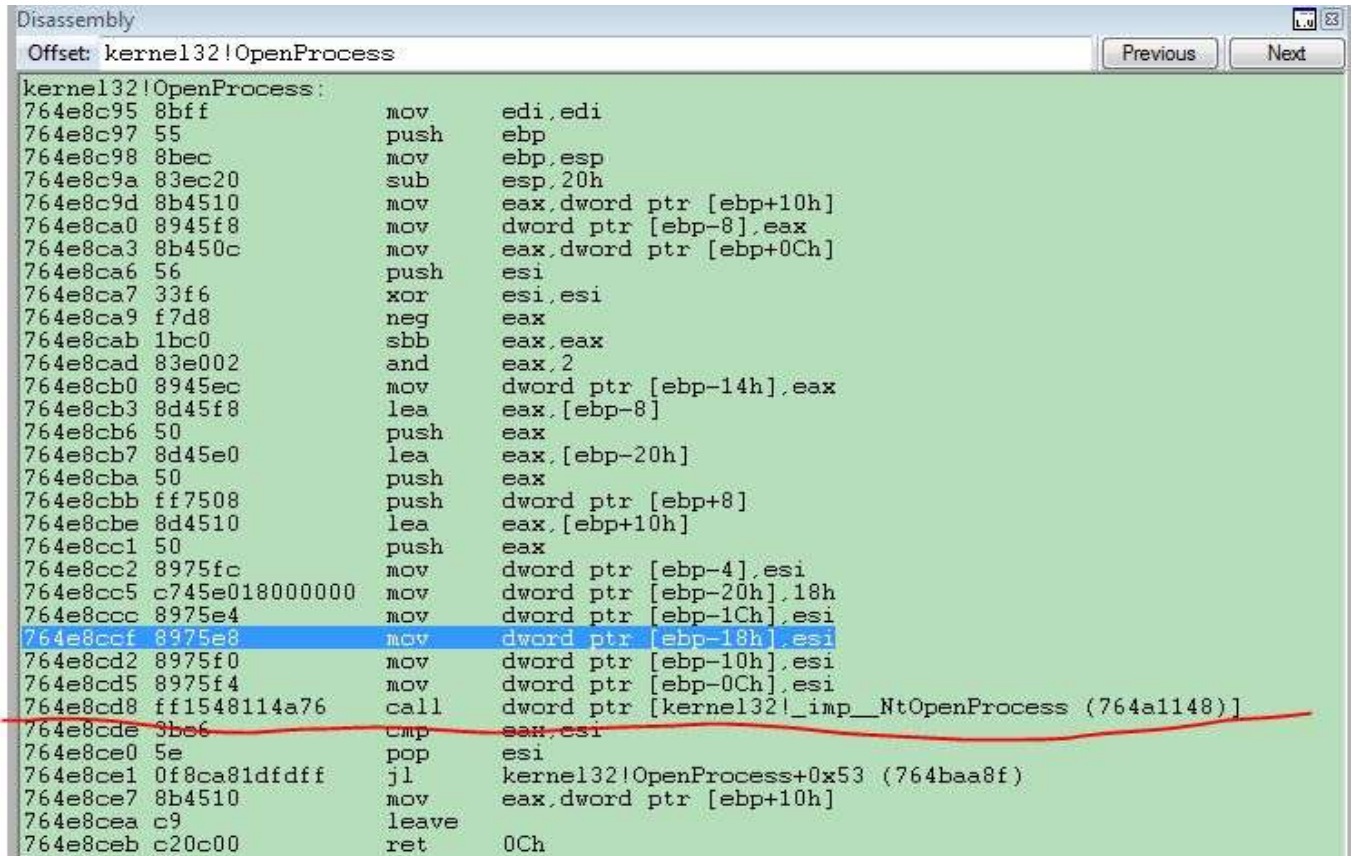
注意位于 764e8ccf 处的那个 call，现在只能看到调用了 kernel32 某个偏移处的地址。

使用命令 .symfix+ d:\Symbols 命令，注意加号要紧靠前面的文本。d:\Symbols 是用来保存下载的符号文件的目录，可以修改成自己需要的路径。再来打开符号路径窗口，我们可以看到调试器自动添加了一些内容：



自己在源码路径中加入这些新的内容也可以实现相同的效果。详细的原理请参考 WinDbg 帮助文档关于符号服务器设置的部分内容。

接下来再次使用 .reload 命令重新加载符号，第一次使用到的符号文件会从网上自动下载下来，所以可能有时候会等待一会。完成之后，可以看到反汇编窗口中出现了新的符号内容：



```
Disassembly
Offset: kernel32!OpenProcess
Previous Next

kernel32!OpenProcess:
764e8c95 8bff      mov     edi,edi
764e8c97 55        push    ebp
764e8c98 8bec      mov     ebp,esp
764e8c9a 83ec20    sub     esp,20h
764e8c9d 8b4510    mov     eax,dword ptr [ebp+10h]
764e8ca0 8945f8    mov     dword ptr [ebp-8],eax
764e8ca3 8b450c    mov     eax,dword ptr [ebp+0Ch]
764e8ca6 56        push    esi
764e8ca7 33f6      xor     esi,esi
764e8ca9 f7d8      neg     eax
764e8cab 1bc0      sbb     eax,eax
764e8cad 83e002    and     eax,2
764e8cb0 8945ec    mov     dword ptr [ebp-14h],eax
764e8cb3 8d45f8    lea     eax,[ebp-8]
764e8cb6 50        push    eax
764e8cb7 8d45e0    lea     eax,[ebp-20h]
764e8cba 50        push    eax
764e8cbb ff7508    push    dword ptr [ebp+8]
764e8cbe 8d4510    lea     eax,[ebp+10h]
764e8cc1 50        push    eax
764e8cc2 8975fc    mov     dword ptr [ebp-4],esi
764e8cc5 c745e018000000 mov     dword ptr [ebp-20h],18h
764e8ccc 8975e4    mov     dword ptr [ebp-1Ch],esi
764e8ccf 8975e8    mov     dword ptr [ebp-18h],esi
764e8cd2 8975f0    mov     dword ptr [ebp-10h],esi
764e8cd5 8975f4    mov     dword ptr [ebp-0Ch],esi
764e8cd8 ff1548114a76 call    dword ptr [kernel32!_imp__NtOpenProcess (764a1148)]
764e8cde 3bc6      cmp     esi,esi
764e8ce0 5e        pop     esi
764e8ce1 0f8ca81dfdf jl      kernel32!OpenProcess+0x53 (764baa8f)
764e8ce7 8b4510    mov     eax,dword ptr [ebp+10h]
764e8cea c9        leave   eax
764e8ceb c20c00    ret     0Ch
```

764e8cd8 处指令中可以看到这是调用了 kernel32 导入的函数 NtOpenProcess。

微软提供的 Windows 符号是我们研究 Windows 实现的必备利器。首先，符号化的名字有助于调试过程中的记忆和对各种信息的识别；其次，通过名字就常常可以猜测出来函数或变量的作用，很大的方便调试。在各种调试应用中，都强烈建议添加微软公共符号的引用。

- 设置环境变量

上面介绍的各种路径都可以通过环境变量来进行设置。将一些常用的路径保存在环境变量中，就可以避免每次在新的工作空间中进行调试时都要重新设置的麻烦。另外，Visual Studio 2008 也共享一些环境变量的设置，这样在使用 IDE 调试的时候也能方便的查看到各种符号了。常用的有下面几个：

环境变量	作用
_NT_SOURCE_PATH = Path	指定包含调试目标的源代码的路径。Path 可以包含后跟一个冒号(:)的驱动器符。用分号分隔多个目录(;)。
_NT_SYMBOL_PATH = Path	指定包含符号文件的目录树的根目录。Path 可以包含后跟一个冒号(:)的驱动器符。用分号分隔多个目录(;)。
_NT_EXECUTABLE_IMAGE_PATH = Path	指定包含二进制可执行文件的路径。Path 可以包含后跟一个冒号(:)的驱动器符。用

	分号分隔多个目录(;)。
<code>_NT_DEBUG_LOG_FILE_OPEN = Filename</code>	(仅 CDB 和 KD) 指定调试器用来记录输出的日志文件。
<code>_NT_DEBUG_LOG_FILE_APPEND = Filename</code>	(仅 CDB 和 KD) 指定调试器用来添加输出的日志文件。新的内容每次会添加到这个文件末尾，而不是覆盖整个文件。

如果设置了符号路径的环境变量的话，可能在初期使用 VS 2008 调试 MFC 这样的有较多导入库的程序时会下载很多符号文件，使得启动调试的速度变慢。不过经过一段时间，大部分需要的符号都缓存到本地之后速度就会快起来。

二、配置日志文件

进行调试时，有时候调试器命令窗口会变得很杂乱，所以常常想用 `.cls` 命令清空它。但是这样会无法再看到之前调试过程中输出的结果。另外，有时候想保存下整个调试过程的详细记录以备后面“回味”。这时，就需要用到日志文件了。可以将调试器命令窗口中出现过的所有内容都自动记录到日志文件中。

创建日志文件：

- (仅 CDB 和 KD) 启动调试器之前，设置 `_NT_DEBUG_LOG_FILE_OPEN` 环境变量。
- 启动调试器时，使用 `-logo` 命令行选项。如 `-logo d:\logs\mylogfile.txt`
- 使用 `.logopen` 命令。如 `.logopen /t d:\logs\mylogfile.txt`
- (仅 WinDbg) 使用 Edit->Open/Close Log File 菜单命令。

将日志添加到已有的文件末尾：

- (仅 CDB 和 KD) 启动调试器之前，设置 `_NT_DEBUG_LOG_FILE_APPEND` 环境变量。
- 启动调试器时，使用 `-loga` 命令行选项。如 `-loga d:\logs\mylogfile.txt`
- 使用 `.logappend` 命令。如 `.logappend/t d:\logs\mylogfile.txt`
- (仅 WinDbg) 使用 Edit->Open/Close Log File 菜单命令，然后选择 **Append**。

关闭日志文件：

- 使用 `.logclose` 命令
- (仅 WinDbg) 使用 Edit->Open/Close Log File 菜单命令，然后选择 **Close Open Log File**。

三、设置工作空间

工作空间(Workspace)是用来保存 WinDbg 中工作环境的工具。例如习惯的窗口布局方式、符号路径、异常处理的设置等等，都可以通过工作空间保存下来，在下次调试的时候就不用再次设置了。

相关的设置都可以通过 WinDbg 菜单来完成，有下面几个：

- **Open Workspace**：这里只能打开自己通过 **SaveAs** 保存的工作空间。

- **Save Workspace:** 按默认的方式保存当前的工作空间。下次再打开相同的调试目标时，就会自动打开这个 **Workspace**。
- **Save Workspace As:** 可以自己设置工作空间的名字，这样就能通过 **Open Workspace** 来手动打开。
- **Clear Workspace:** 可以选择保存工作空间时要保存哪些设置。
- **Delete Workspace:** 删除当前保存的工作空间。这里可以查看到所有默认保存和另存为的工作空间，用来进行清理是很方便的。
- **Save Workspace in File** 和 **Open Workspace in File:** 将工作空间保存到文件或者从文件打开。可以把自己的工作空间保存下来，这样通过 U 盘之类的就能在多台机器之间方便的使用相同的设置了。

在没有调试目标的时候调整 WinDbg 的窗口布局等等设置的话，会保存为默认的工作空间。下一次打开新目标的时候，就会使用这个设置。通常我们可以设定一个默认的工作空间，然后为各个单独的任务保存另外的设置。