关于 Windbg 的 学习笔记

1. ctrl+break 在windbg中表示中断 其中break 是键盘的pause

也就是scrlk右边的键，也是截屏右侧2个位置的键

1. bp下断点的项目如果存在带空格的字符串，则

2017-04-14 windbg项目测试！main 无法中断

但是如果是正常的标准abc程序使用

TestA！main则会在main方法中断

值得注意的是 如果存在exe的路径中包含中文字符串，则使用bl查看 exe路径的使用可以正常解析，也就说可以使用中文，但是不能有空格

1. 使用bp设置断点，使用bc删除断点，bc 删除断点 后面的参数是使用 bl命令查看断点的索引。 如 bc 0
2. 使用 bp TestDebug1.exe!CTestClass::SetChar

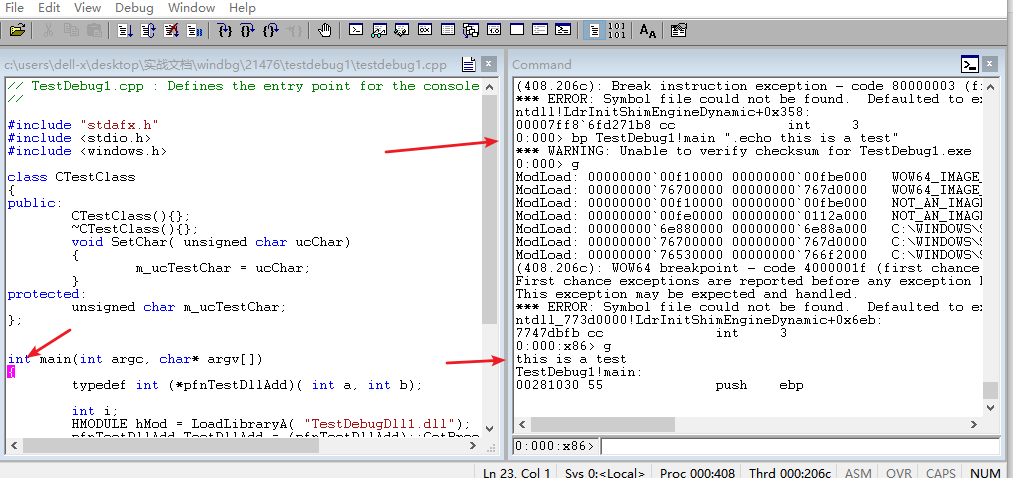
对类内的对象进行断点操作

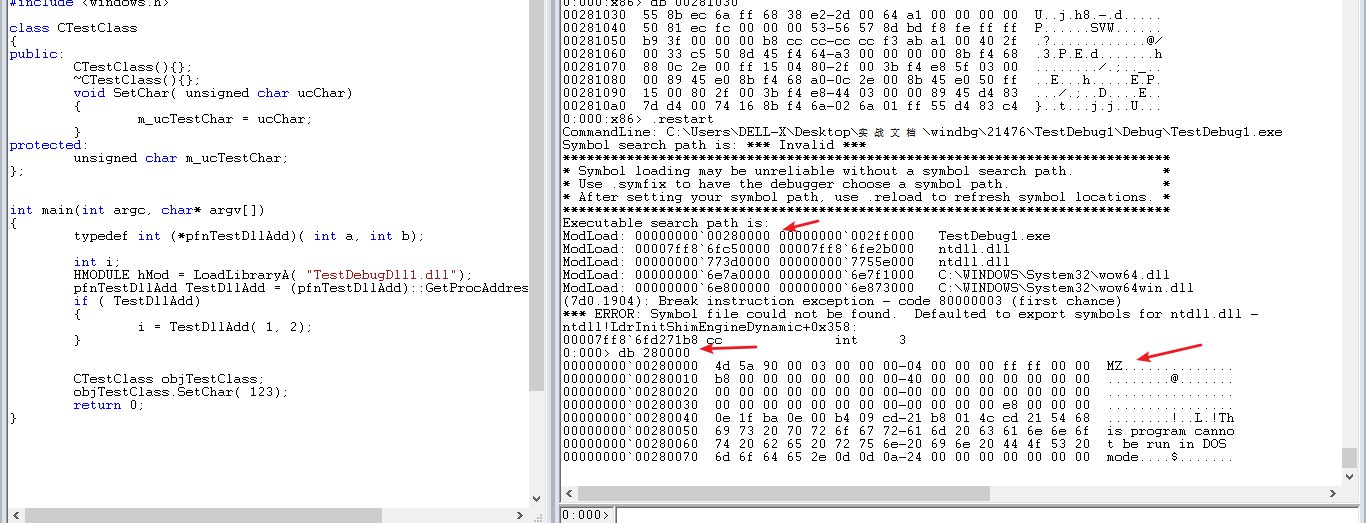
1. 当使用 ba 设置访问断点，也就是内存断点的时候

如 ba w4 @@C++(&i)

值得注意的是

1. w4和后面的参数传是有空格的
2. 如果需要对这个内存地址设断点，则先要在这个程序的main方法处设断点。要不然会提示找不到参数的地址
3. 内存断点设置完成后，使用 g命令执行 下一个断点。

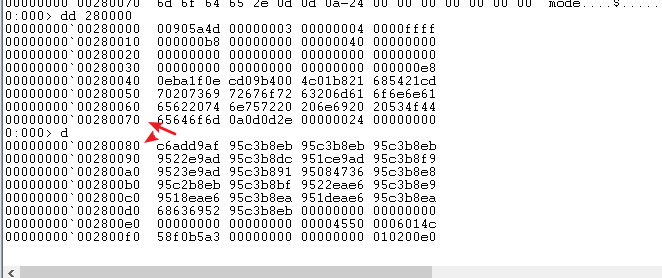


1. 运行到指定断点的时候也可以输出一些内容。
2. 使用g命令可以运行到下一个断点。也就是说g是触发断点的作用。
3. 

最上边的箭头指明了程序的基址，每次都会变

中间的箭头指明 用来查看数据

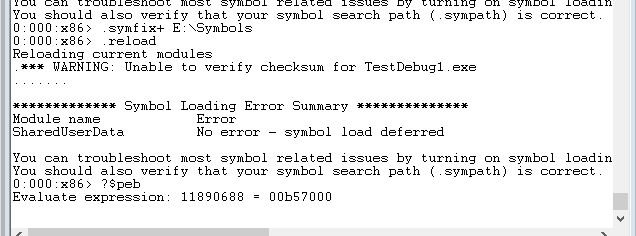
最下面的箭头指明 pe格式的开头，表明了基址的位置。

1. db debug byte 查看一个字节， dd debug dword查看一个双字
2. 

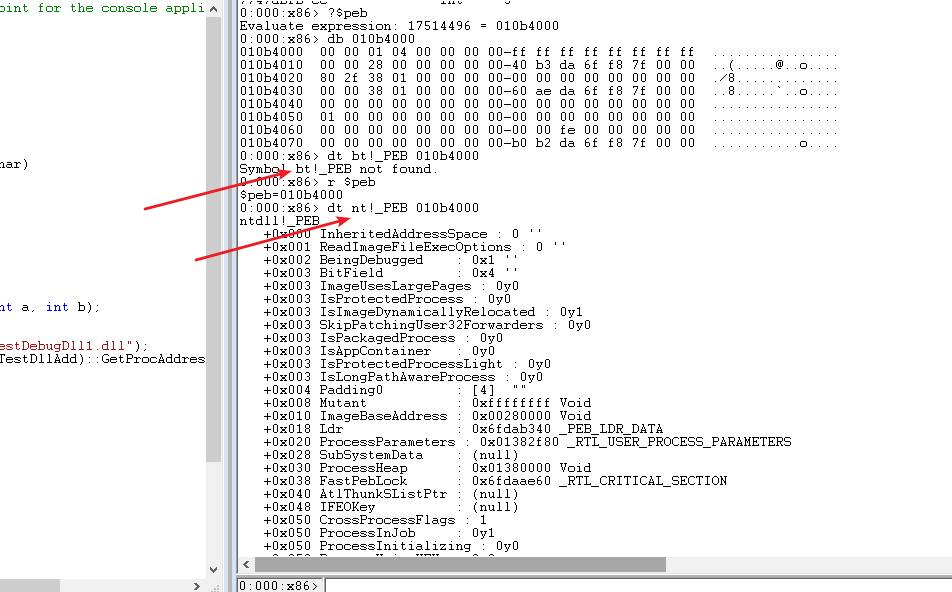
D命令会程序执行上一个命令；

1. 使用eb 00400000 ‘a’ ‘b’ ‘c’ ‘d’可以向指定的位置写入输入数据

但是值得注意的是

1. 对于写入数据的位置程序应该具有内存访问权限
2. Abcd之间使用空格间隔。
3. 使用？ Arg的时候应该注意和 在ba @@c++（&i） 存在同样的问题，应该进入主程序，才能使用，所以应该在主程序处断点。
4. 

上图演示了 载入符号集的方法

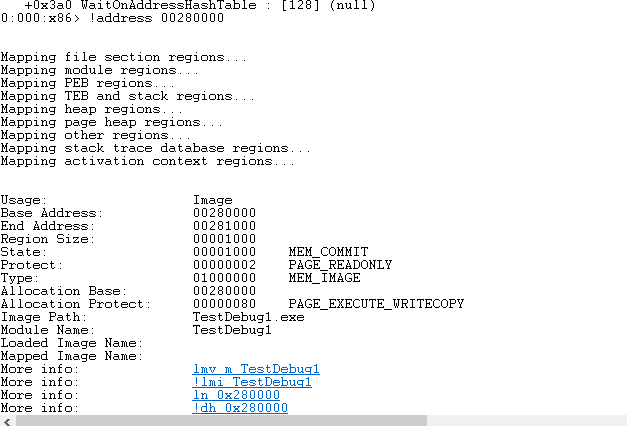
1. 

查看进程环境块

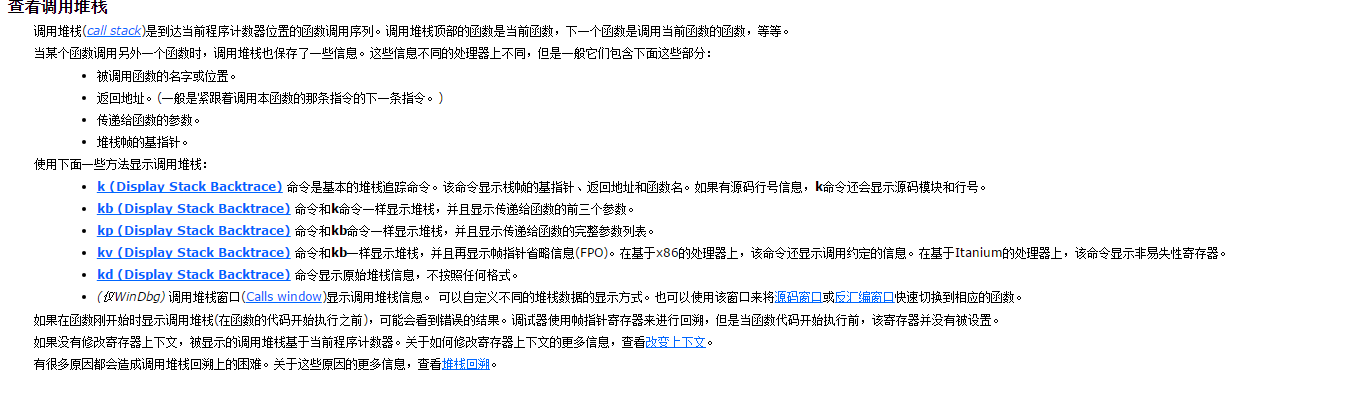
R $peb

0x40000000

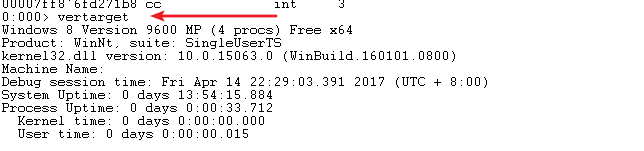
命令是dt nt！\_PEB 0x4000000

15. 

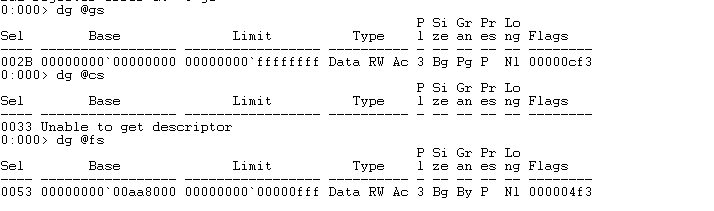
查看指定处的地址属性， 这里地址使用peb中 ImageBaseAddress得到。

1. 

显示堆栈不同命令的区别 kv kd k

1. 

查看操作系统相关信息 vertarget

1. 

使用windbg调试远程计算机的方法

1. 在windbg的属性中设置目标"C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64\windbg.exe" -b -k com:pipe,port=\\.\pipe\com\_1,resets=0
2. 如果被调试的目标机器是xp则需要修改boot.int文件，打开boot.ini的方法是 win+r 输入c：/boot.ini 则会使用记事本程序打开

此外，被调试的机器需要重启后 才能生效。

需要注意的是

1. boot中可能存在和输入相似的字符串，但是不要修改，只要在最下边加入调试字符串即可
2. 重启后，需要在选在window窗口中选择调试模式启动windows，类似与window和dos共存的窗口一般
3. 调试时需要使用 pipe管道，串口等辅助，虚拟机如果没有添加串口则需要手动添加串口，添加的方法就像添加硬件设备一样。
4. 调试的时候，应该先打开windbg软件，windbg界面会出现wait for connecting等 等待连接的字符串，这个进入调试window，

可能出现的问题

Windbg出现闪退，解决的方法是重启下windbg则正常恢复调试连接，

此外。Windbg需要中断windows才能调试，所以应该注意中断快捷键和中断按钮的使用

此外，如果想要在调试过程中，使用调试windows应该使windows处于运行状态，快捷键是F5

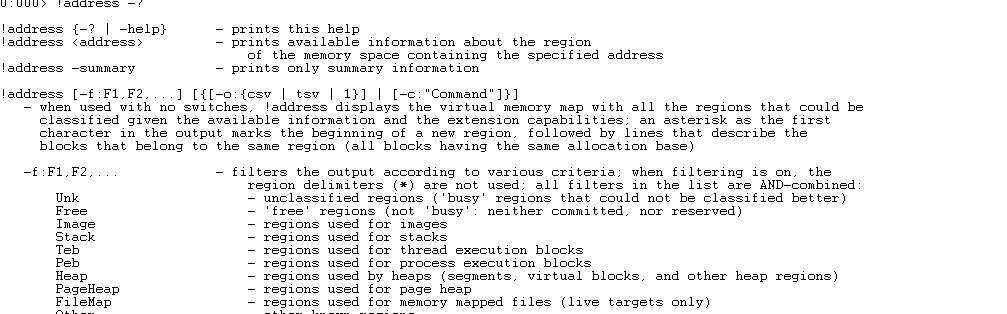
另外，值得注意的是windbg不能向普通应用软件一般，设置管理员权限。

1. Windbg设置管理员权限的方法是 右键 属性 快捷 高级 即可看到设置管理员权限的提示了。
2. Windbg的一些命令只能是远程 调试时使用 如process，cpuinfo等

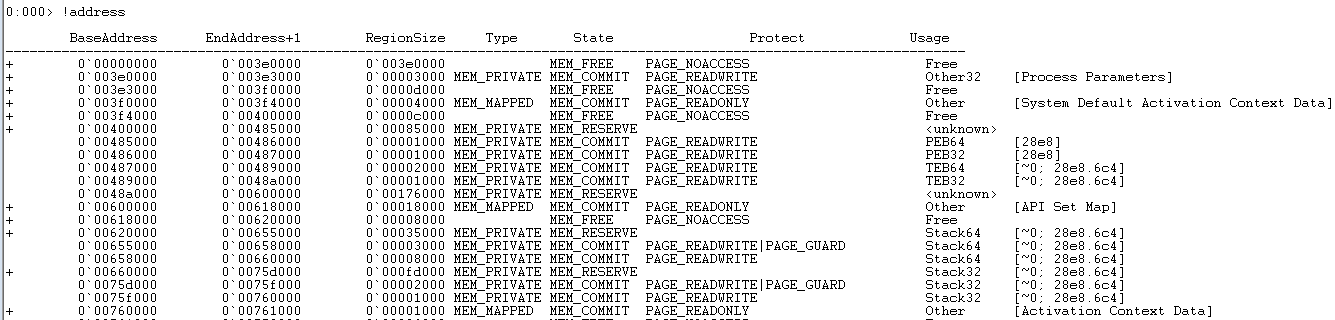
Windbg 的命令的具体用法使用可以使使用 command -？显示

不过windbg有些命令分为用户态和内核态之分

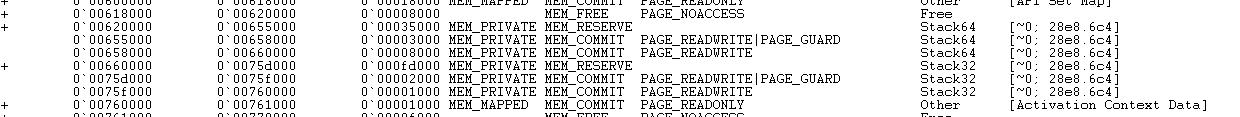
例如 ！address -？



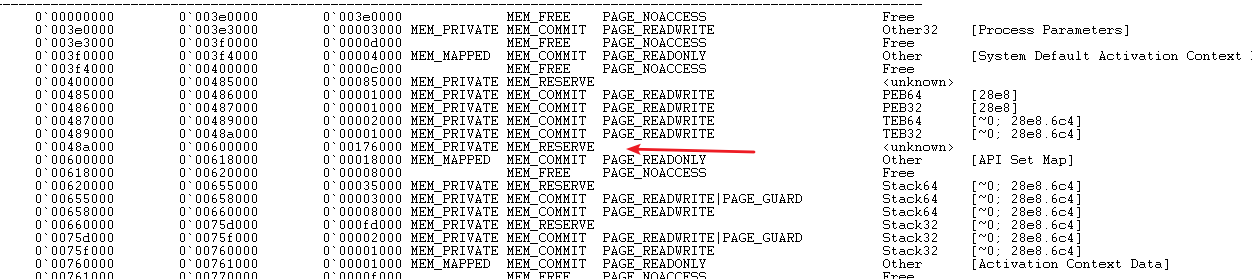
主要说明的是 因为 address是用户态的命令，此外使用address的时候需要使用命令行，所以应该载入一个程序 ，然后使用命令行调试。



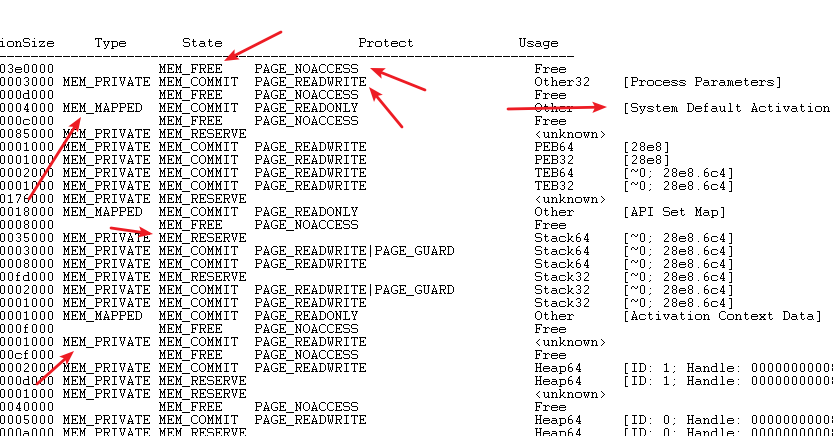
上图为address的执行结果，需要注意的是 可能没有和列标题对齐



栈分为32位和64位



内存 存在恢复区，用来恢复数据，和路由器刷机的时候，路由器内部的分区有些相似，路由器内存的rom分为主引导分区，恢复分区等，



从上图的箭头可以看出，内存的属性，如mem\_mapped说明内存的结构采用键值对集合，但是其他memeory type 就是men\_privaty

从尾部的usage可以看到内存的描述。用来描述这段内存的作用。

Windbg调试驱动sys程序

1. 驱动程序和dll或者是其他的exe程序一样，都是有一个主入口点的。 主入口点的名字叫做DriverEntry
2. Windbg中有专门的插件调试驱动程序 KMDmanager.exe

Kernel-mode driver manager

1. 如果之前使用vs6.0开发驱动程序，当使用vs2013的时候，就会出现，字符编码的问题 unicode。解决方法是安装一个插件
2. 调试驱动程序的时候，如果不在本机开发的程序，那么调试的时候应该设置symbol路径

设置路径的方法是 .sympath

值得说明的是

可以设置多个符号路径的路径值，不过多个路径之间需要使用;分号隔开

此外，windbg中的命令开头是 . 点号 或者是 ! 叹号作为开头。

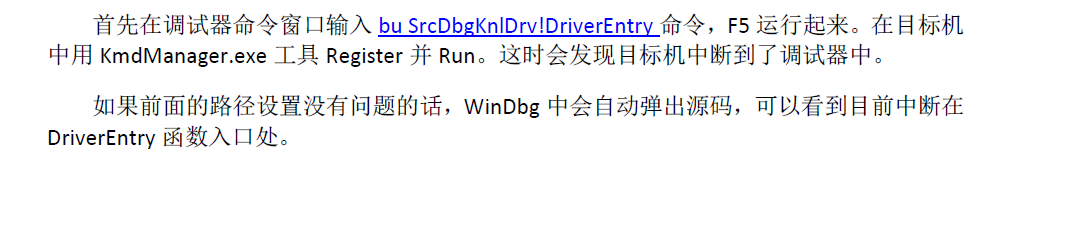
.Sympath !address

1. 驱动程序会访问注册表，所以值得注意的是可以利用注册表调试特定的驱动程序。

此外值得注意的是

使用kernel-mode driver manager调试驱动程序的时候应该注意

一定要按照 regesiter run stop unregetiser 的步骤，这样做是保证可以关闭所有的句柄，删除注册表对应的关键键值，否则只有重启系统才能卸载驱动程序。



调试驱动程序的步骤。



Kernel -mode driver manager 程序截图

Dispatch 例程是驱动程序响应用层请求的地方，通常来说，驱动程序和应用程序的各种交互就是通过Dispatch例程实现，因此，要跟踪应用层请求处理的情况，可以在这些Dispatch例程上设置断点。

值得说明的说 驱动程序有两个值得下断点的程序

1. DriverEntry 2.Dispatch
2. 使用 devhandles命令应该注意的问题

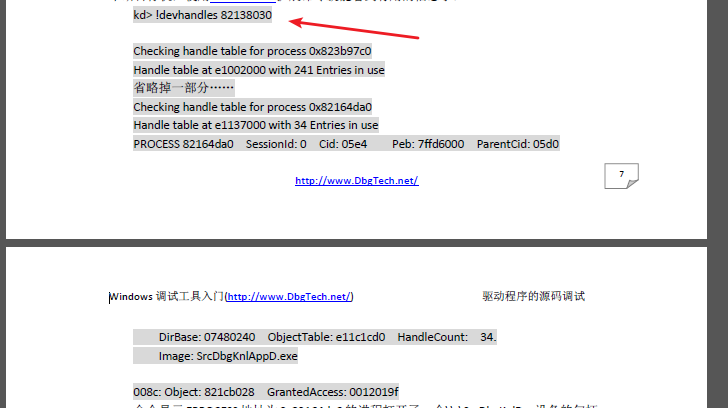
使用devhandles查看指定地址的句柄情况的时候

1. 首先应该使用 ！drvobj drivername 获取对应驱动程序的内存地址



1. 使用devhandles +地址 828138030 即可获取到 哪些程序使用了驱动程序的句柄。

值得注意的是 devhandles 后边的地址是驱动程序的地址



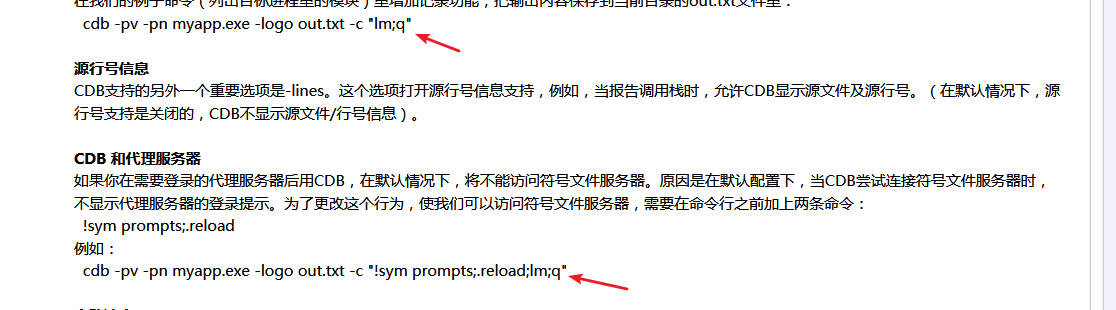
如果想要查看具体进程的信息，可以使用 process processaddress

使用bc \*可以清除所有的断点。

1. 此外，调试驱动程序的时候，首先应该使用的是 analyze -v进行自动化分析，如果没有引入符号表和源文件，默认可以分析出来驱动程序的文件名 mydriver.sys

当引入完符号表，应该使用 reload命令重载程序。

当时cdb windbg 命令行模式的时候

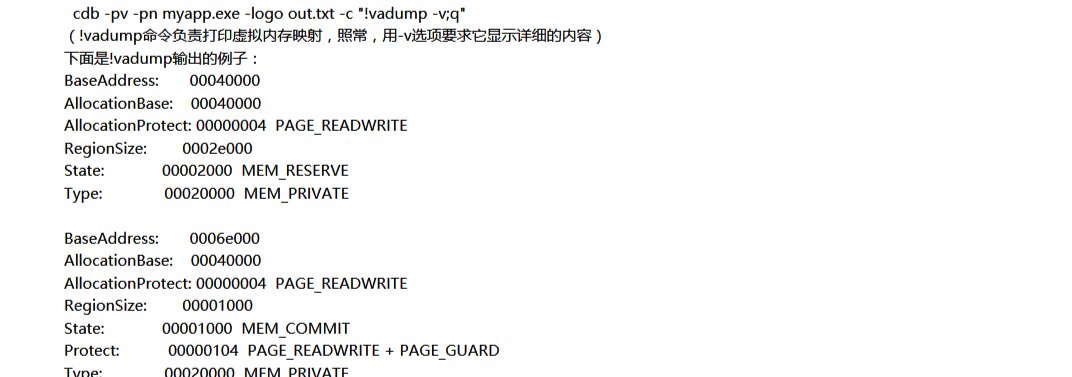


如果设置参数，则参数的位置在每行最后的尾部，并且使用双引号括起来。

使用cbd程序的时候，命令的结尾使用 q结束

调试cdb的时候，使用命令 kb将显示所有的堆栈信息。使用kf将显示每个函数在调用堆栈上占用的字节数。Kf默认显示调用堆栈上最后的20帧，使用~\*kf1000，可以显示1000帧，可以使用~\*kf 显示所有线程的调用栈。可以使用~~[tid] kf显示对应kid进程的堆栈情况。

Cdb中使用vadump 可以查看虚拟内存映射，使用 -v参数可以查看完整的内存映射

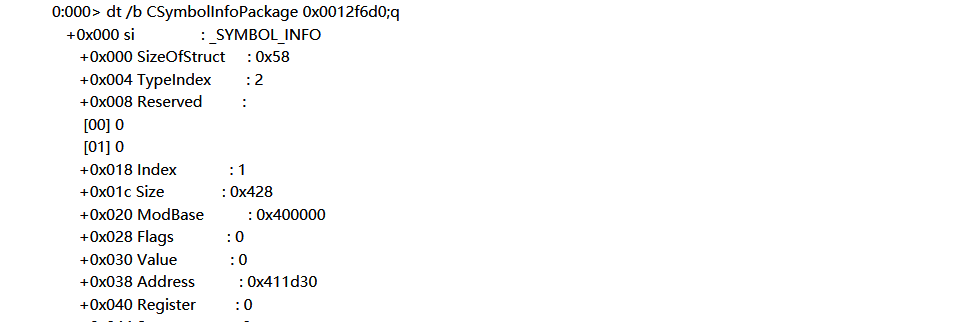


和xp和2003中，使用address可以查看虚拟内存映射，

X Module！sysmbol可以定位，在对应模块中sysbol所在的所有位置 此外 x命令支持通配符 \* ，

Ln address ln命令用来反向解析 地址成汇编格式

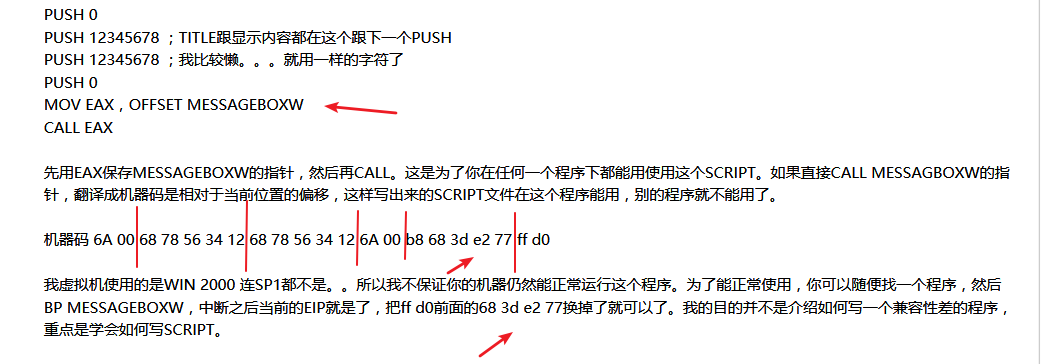
Dt -b typename address 可以 以基地址为基础，根据时期参数情况动态解析 数据结构



MOV EAX，OFFSET MESSAGEBOXW  
CALL EAX

关于为什么不能直接使用 call messageboxw

因为这个程序是汇编程序，需要经过编译才能运行，如果使用call messageboxw这种写法，那么就是调用本机的messagebox，作为编译时候的机器码填充，其他机器如果messageboxw地址不同，那么则无法运行。



如果想要直接call eax 则只要将ff d0 即 call eax的之前的代码 mov eax，offset messageboxw的机器码替换成 nop

即可，注意的是 机器码的字节对齐的问题，因为在程序运行之前pe结构中会有一个数据结构表示代码区域的大小，所以这里要想程序可用应该使用 5个90替换 B8 68 3D E2 77



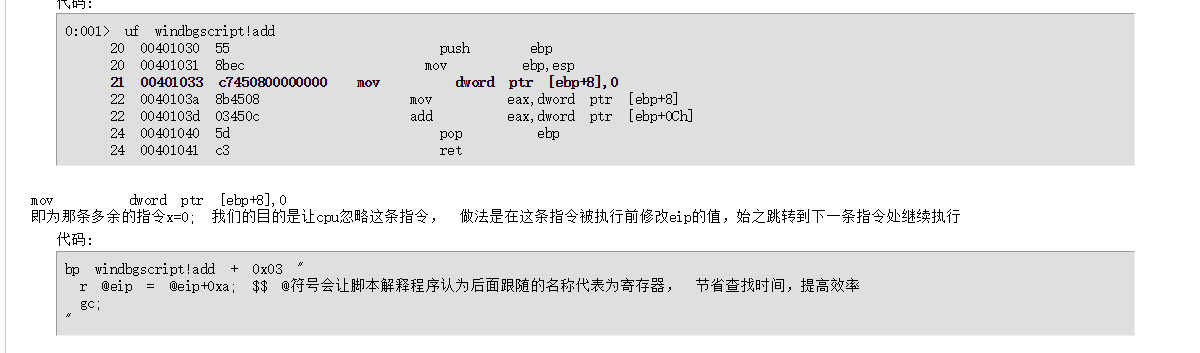
表示在t1表示的地址处，写入的长度为20个字节数据。

写入的数据是 6A 00 B8 68 3D E2 77 FF D0 9个字节的数据

也就是后边还有11个字节没有覆盖写入。

bp messageboxw ;    $$第一个断点，断点ID为0  
bp getcommandlinew; $$第二个断点，断点ID为1  
r $t10 = $bp0;             $$将第一个断点的地址转存$t10  
r $t11 = $bp1;             $$将第二个断点的地址转存$t11  
bc \*;               $$清除所有断点。  
  
这段代码运行之后MESSAGEBOXW的地址便存于$t10中，而$t11里面的则是getcommandlinew的地址。这里要说明，断点用完要释放，否则不好估计断点的ID，此外在内核模式中，最多只允许32个断点。

使用SCRIPT文件的命令有4个"$<","$$<","$><","$$><"，他们的区别就是有没有空格或者换行符的限制。使用$$><没有任何限制，这样可使代码更具可读性。要使用附件中的SCRIPT请使用$$><指令。  
  
例如你可以用下面指令访问在D盘下的helloworld.txt。  
  
$$><d:\helloworld.txt



bp windbgscript!add + 0x03 "

 r @eip = @eip+0xa; $$ @符号会让脚本解释程序认为后面跟随的名称代表为寄存器， 节省查找时间，提高效率

 gc;

值得说明的是

1. windbg中使用 $$作为注释符号
2. @eip 表示的是 eip这个寄存器，可以取值也可以赋值
3. Bp windbgscript！add +0x03的解释是

Windbgscript是被调试的程序的名称

Add是中断的方法的名称

0x03 表示的是偏移的地址

结合上下文的意思就是

在 21 00401033 c7450800000000 mov dword prt [ebp+8],0这行下断点，然后修改 eip 指令寄存器的值，执行下一条指令的地址是当前的指令地址向后偏移 a 10 个字节的位置

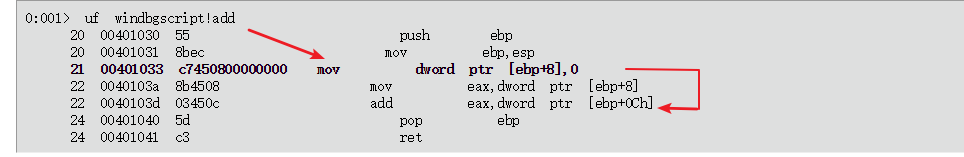
1. 关于 0xa的解释



这两条指令的含义是将 0复制给eax 寄存器，

由于eax本身就是0，所以赋值有些重复，因此调试这两条指令，达到优化代码的目的。

这两条指令的 机器字节码是 20个 16进制位 ，也就是 10 个字节，16进制表示就是 0xa；



实际上想要表示的就是这样一个效果。

值得说明的是

1. windbg中使用 s 进行字符串查找

值得说明的

S -a 0012ff40 L20 “hello”

-a 指定地址， L20 因为查找的最小长度是20，如果想要查找的字符串长度小于20，也要写20

即搜索的区间是 0012ff40---0012ff60 60-40=20

字符串查找的三种等价形式

1.

S 0012ff40 L20 ‘h’’e’ ‘l’’l’ ‘o’

字符查找，使用单引号括起来，并且使用空格隔开

2.

S 0012ff40 L20 48 65 6c 6c 6f

Ascii码查找

3.

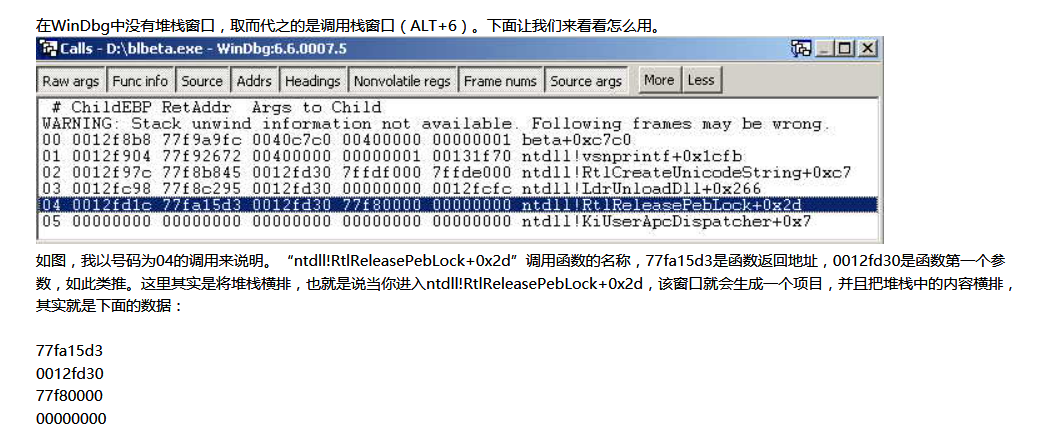
S -a 0012ff40 L20 “hello”

-a参数和双引号的使用。

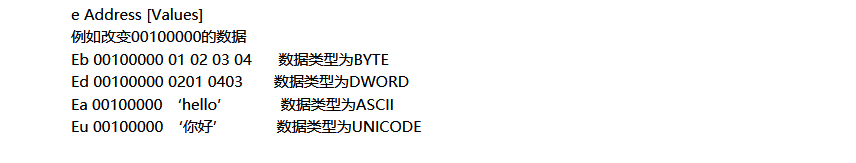
-a 参数表示 使用 ascii方式查找

-u 参数表示使用unicode方式查找

Windbg中堆栈参数的解释



Windbg 改变内存数据



TLS 是 TLSCALLBACK 的简称，TLSCALLBACK的含义是 方法回掉

Tls说明

1. tls方法中可以访问方法外的变量 也就是 esp+8 ，esp+x表示访问方法外。
2. tls方法可以帮助父过程 平衡堆栈空间。
3. 访问方法外的时候应该注意 ss堆栈寄存器 dword ptr ss:[esp+8]表示一个32的值，一个地址

kp 5

显示调用栈中前5个函数以及他们的参数.

kb 5

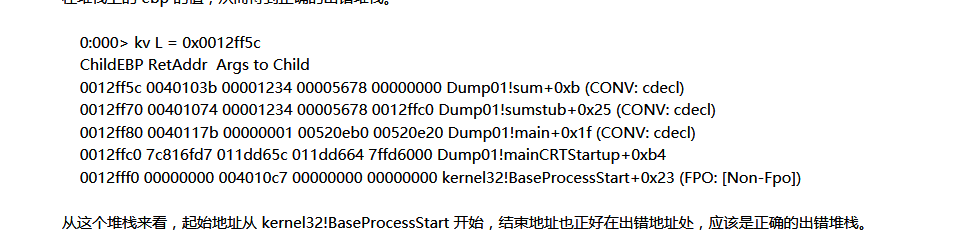
显示调用栈中前五个函数以及他们的前三个参数.

kf 5

显示在调用栈中五个函数所使用的栈的大小.

每个栈帧所占的空间使用量的计算方法是: 将当前函数的栈基指针与在函数中调用的任何一个函数栈基指针相减

k 关于堆栈参数问题的解释



以第一行数据为例



0012ff5c 表示当前堆栈在内存中的地址

0040103b 表示当前函数执行完后的返回地址

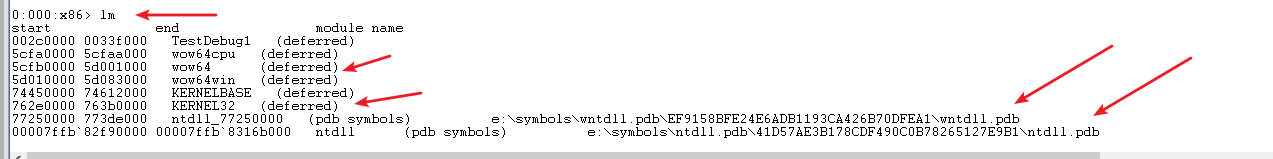
00001234 表示当前函数sum 的第一个参数 即0x1234

00005678 表示当前函数sum的第二个参数 即0x5678

00000000 表示当前函数的第三个参数由于sum只有两个参数所以显示为0

Windbg中自动生成dump文件的方法

Windows Registry Editor Version 5.00  
  
[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug]  
"Auto"="0"  
"Debugger"="\"D:\\WinDBG\\ntsd.exe\" -p %ld -e %ld -g -noio -c \".dump /ma /u d:\\dbgdmp\\dmp.dmp; q"  
"UserDebuggerHotKey"=dword:00000000



1.使用 lm命令可以列出 当前进程所有装在的模块

当设置完symbol的环境变量之后，可以使用lm命令，验证pdb是否导入进来。

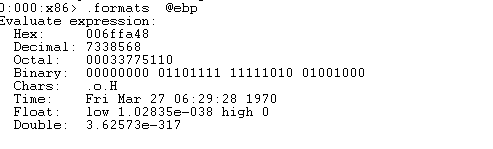
2.！process命令是联机调试的时候使用的，在本机调试时使用会出现

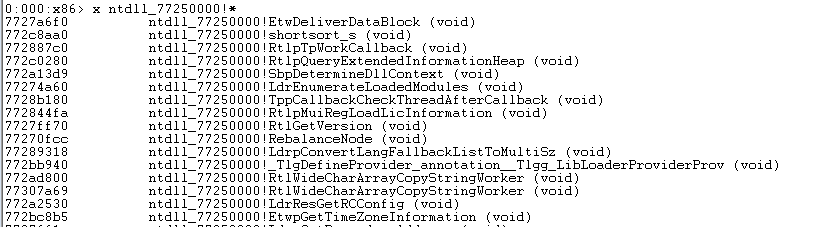
No export process found

1. windows里面的各种数据结构通常采用双向链表连接，所以使用！list查看非常方便。



查看对应寄存器的信息，格式化输出，多种格式输出。





使用X 查看模块的时候应该注意的是

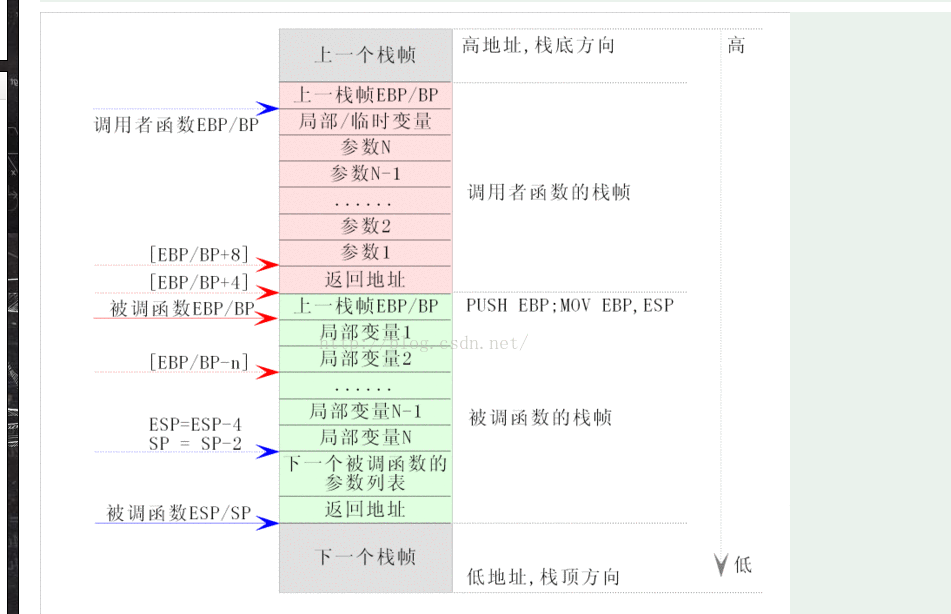
1. 必须写全！开头的名字
2. 可以使用ln查看最近的模块
3. Ntdll的模块的名字ntdll\_77250000 的名字后边是地址

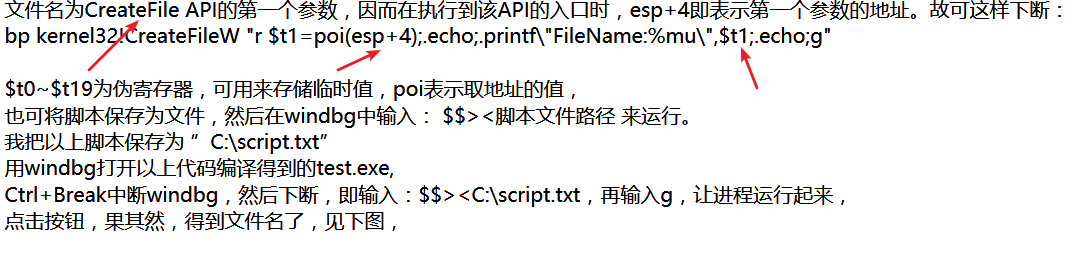
Windbg中poi的作用是 解析指针指向的值

相当于c++中 \*p=&d 中的 \*号

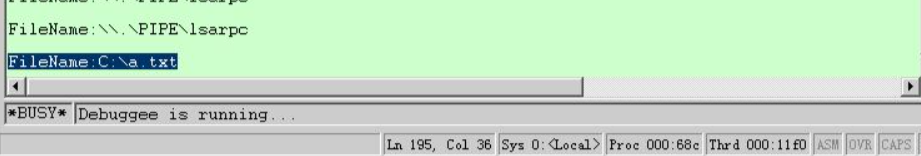


汇编语言中 ebp+4的含义是 当前函数的返回地址





利用api信息获取文件名，因为文件名是createfileapi的第一个参数



关于栈的总结

Ebp表示当前函数栈的基址

Ebp+4 表示函数的返回地址，也就是函数返回时要执行汇编指令的地址

如果方法的参数采用从右向左入栈，由于栈的方向是从高地址往低地址方向运动

Push 右 esp=esp-4；

Push 左 esp=esp-4；

Call function 这里隐含着将返回的地址压栈

Push ebp 保留调用方法前的栈环境，体现了通过栈空间保留现场

Mov ebp，esp 将当前的栈顶作为调用function的栈底

如果function 想要调用传入的参数

Mov eax，dword ptr [ebp+8] 访问第一个参数，如果第一个参数是一个int 类型，也就是说，它存在栈中，而栈中的数据是16进制表示的，那么使用的时候就会做16进制转换成10进制。

如果第一个参数是一个引用类型，那么就会根据这个地址，找到内存中的应用地址空间，由于使用内存的时候，根据使用的不同，不同的变量会分配到不同的地址空间，也就是说，数组站一块，对象站一块，

所以方法就会根据当前方法的内存标记，正确的识别出正确的数据结构，关于内存标记的说明，上文已经分析了。

Mov ebx，dword ptr [ebp+0c] 表示访问第二个参数。

此外，值得说明的是

堆栈平衡的问题。方法调用前后，的堆栈一定是平衡的，也就是说方法调用前后，栈的高度是不会改变的。

Ret 指令 会从栈顶 取一个栈帧 32位数据，作为返回地址，

Ret 16 会将栈顶指针，下压 16进制 16 10进制 22 个，然后再取栈顶 栈帧 作为返回执行的汇编指令的地址。