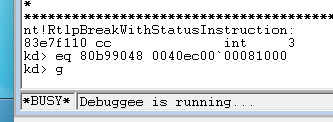
### 2.请用调用门 修改内核高2G的所有物理地址U/S位为1，然后返回R3的时候 随意访问一个高地址，打印出结果，修改地址 从 0x80010000-0xFFFF0000

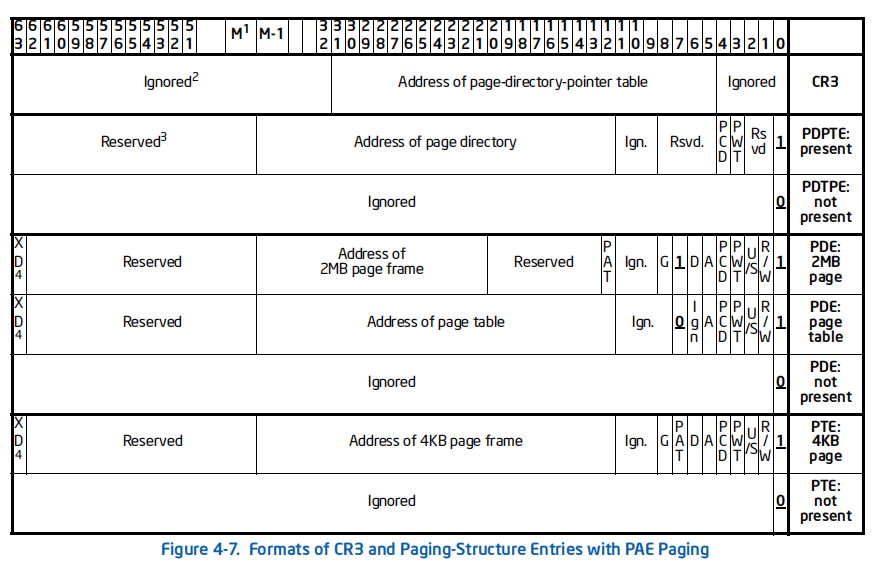
在GDT 48处整一个提权调用门

eq 80b99048 0040ec00`00081000



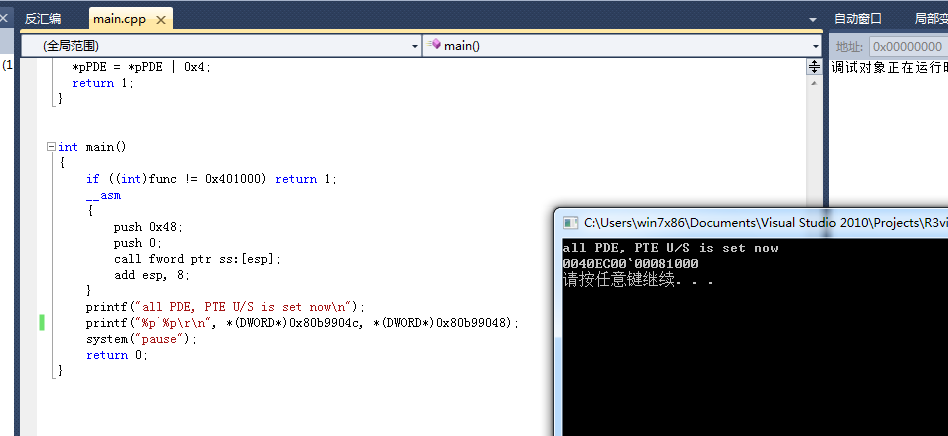
新建项目，关闭增量链接和随机基址，裸函数在 0x401000

然后遍历线性地址，从 0x80010000 到 0xFFFF0000，获取PDE,PTE，如果这个线性地址是有效的，那么把它的PDE, PTE的U/S位改成1



完整代码如下：

|  |
| --- |
| #include <Windows.h>  #include <stdio.h>  bool \_\_stdcall SetPageUS(unsigned int VirtualAddress);  \_\_declspec(naked) void func()  {  \_\_asm  {  pushad;  pushfd;  push 0x30;  pop fs;  //cli;  mov esi, 0x80010000;// 到0xFFFF0000  flag1:  push esi;  mov eax, SetPageUS;  call eax;  add esi,0x1000;  cmp esi, 0xFFFF0000;  jbe flag1;  //sti;  popfd;  popad;  push 0x3B;  pop fs;  retf;  }  }  bool \_\_stdcall SetPageUS(unsigned int VirtualAddress)  {  unsigned int \*pPDE;  unsigned int \*pPTE;  pPDE = (unsigned int \*)(((VirtualAddress >> 18) & 0x3FF8) - 0x3FA00000);  if ((\*pPDE & 1) == 0)  {  return 0;  }  pPTE = (unsigned int \*)(((VirtualAddress >> 9) & 0x7FFFF8) - 0x40000000);  if (!((\*pPTE & 1) != 0 && (\*pPTE & 0x80) == 0))  {  return 0;  }  \*pPTE = \*pPTE | 0x4;  \*pPDE = \*pPDE | 0x4;  return 1;  }  int main()  {  if ((int)func != 0x401000) return 1;  \_\_asm  {  push 0x48;  push 0;  call fword ptr ss:[esp];  add esp, 8;  }  printf("all PDE, PTE U/S is set now\n");  printf("%p`%p\r\n", \*(DWORD\*)0x80b9904c, \*(DWORD\*)0x80b99048);  system("pause");  return 0;  } |



在R3成功读取到GDT 48的内容。

这里有一个注意点是裸函数里的循环要用非易失寄存器esi保存当前地址，用易失寄存器如ecx是不行的。

### 思考题 利用调用门 如何实现访问其他进程的内存

原理是把自己的CR3换成别人的。但是把CR3换掉之后，自己的进程就无法正常访问自己的内存了。因此可以弄一个函数来读写其他进程的内存，每次读写前把CR3换成别人的，读写完再换回自己的。

新建项目，关闭随机基址，增量链接，创建带两个参数的提权调用门。

eq 80b99048 0040ec02`00081000

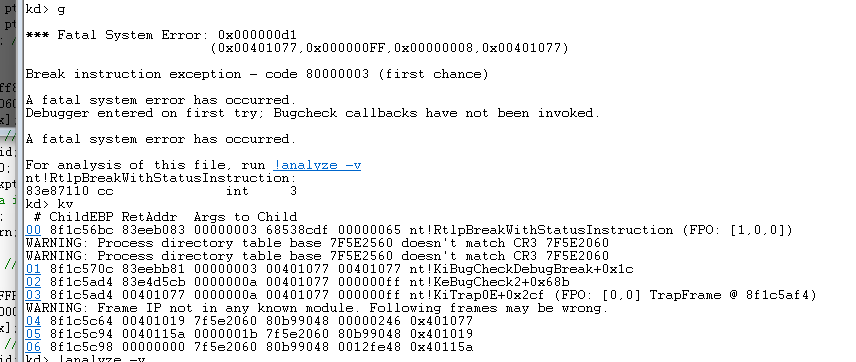
参数1是要读的进程的CR3，参数2是要读的线性地址VA。提权后，切换CR3，通过PDE,PTE检查VA是否有效，有效就读，无效就返回0.为了避免操作过程中切换进程/线程，我们在裸函数执行过程中关闭中断。

缓存的问题？

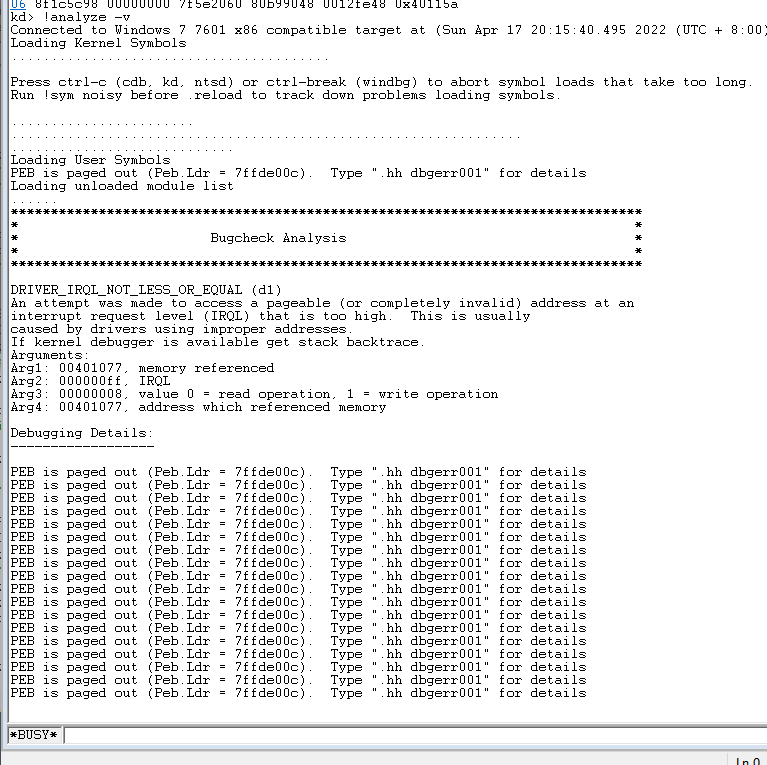
代码如下：

|  |
| --- |
| #include <Windows.h>  #include <stdio.h>  DWORD g\_result;  int \_\_stdcall read4byte(DWORD dwCr3, DWORD va);  \_\_declspec(naked) void func()  {  \_\_asm  {  pushad;  pushfd;  push 0x30;  pop fs;    mov ecx, dword ptr ss:[esp+0x2c]; // cr3  mov edx, dword ptr ss:[esp+0x30]; // va  push edx;  push ecx;  mov eax, read4byte;  call eax;  mov dword ptr ds:[g\_result], eax;  popfd;  popad;  push 0x3B;  pop fs;  retf 8; // 2 param  }  }  int \_\_stdcall read4byte(DWORD dwCr3, DWORD va)  {  int result = 0;  \_\_asm  {  pushad;  pushfd;  cli;  mov edi, dwCr3; // other cr3  mov esi, cr3; // old cr3  mov edx, va; // va  mov ecx, va;    mov cr3, edi; // after cr3 changed, we can visit our memory    shr ecx, 18;  and ecx, 0x3ff8;  add ecx, 0xc0600000;  mov ecx, [ecx]; // ECX=PDE  test ecx, 1; // PDE.P  jz flaginvalid; // PDE.P == 0  cmp ecx, 0x80; // PDE.PS == 1  jnz flagcheckpte;  // we know va is valid  mov eax,[va];  jmp flagreturn;  flagcheckpte:  mov ecx,edx; // ecx = va  shr ecx, 9;  and ecx, 0x7FFFF8;  add ecx, 0xc0000000;  mov ecx, [ecx]; // ecx = PTE  test ecx, 1; // PTE.P  jz flaginvalid; // PTE.P == 0  test ecx, 0x80;  jnz flaginvalid; // PTE.PAT == 1, invalid  // now we know va is valid  mov eax, [va];  jmp flagreturn;  flaginvalid:  xor eax,eax;  flagreturn:  mov cr3, esi;  mov result, eax;  sti;  popfd;  popad;  }  return result;  }  int main()  {  char bufcode[] = {0,0,0,0,0x48,0};  if ((int)func != 0x401000) return 1;  g\_result = 0;  \_\_asm  {  push 0x80b99048; // virtual address  push 0x7f5e22a0; // explorer.exe cr3  call fword ptr bufcode;  }  printf("g\_result: %p\n", g\_result);  system("pause");  return 0;  } |

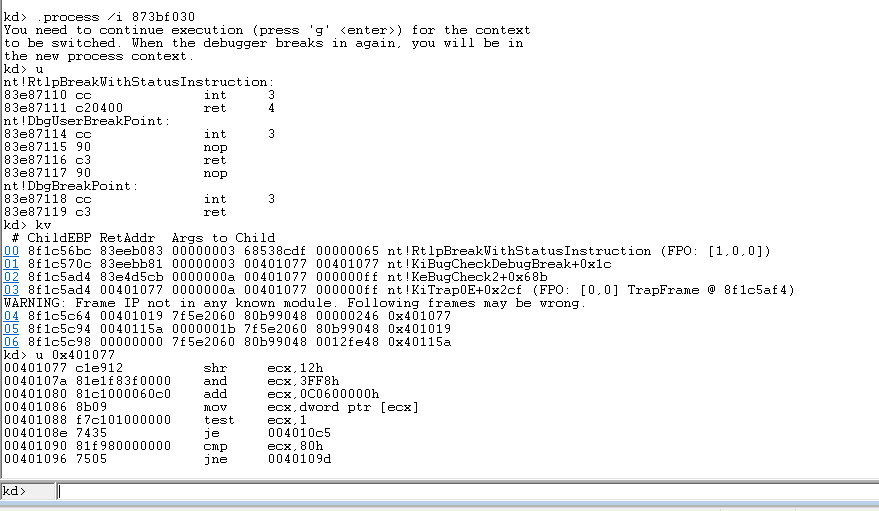
执行出错了，如图：



是切换到新CR3后出错的



附加回自己的进程，kv看看



发现在切换CR3后执行下一条指令时出错了，猜测原因是因为切换CR3后，eip仍然是旧进程的值，所以一执行就出错。