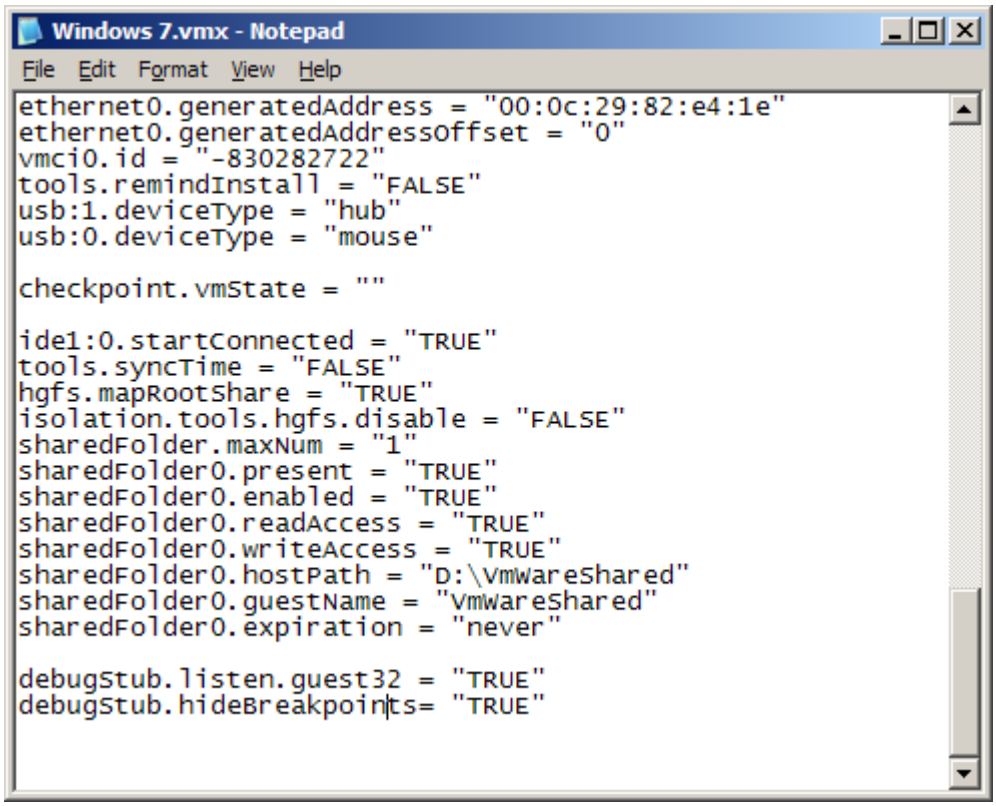
Debugging Windows kernel under VMWare using IDA's GDB debugger

Current versions of VMWare Workstation include a GDB stub for remote debugging of the virtual machines running inside it. In version 5.4, IDA includes a debugger module which supports the remote GDB protocol. This document describes how to use it with VMWare. As an example, we'll debug a Windows kernel.

Preparing VM for debugging

Let's assume that you already have a VM with Windows (32-bit) installed. Before starting the debugging, you can copy the kernel for symbol retrieval later. Copy ntoskrnl.exe or ntkrnlpa.exe (depending on the VM configuration) from Windows/sytem32.

Now edit the VM's .vmx file to enable GDB debugger stub:



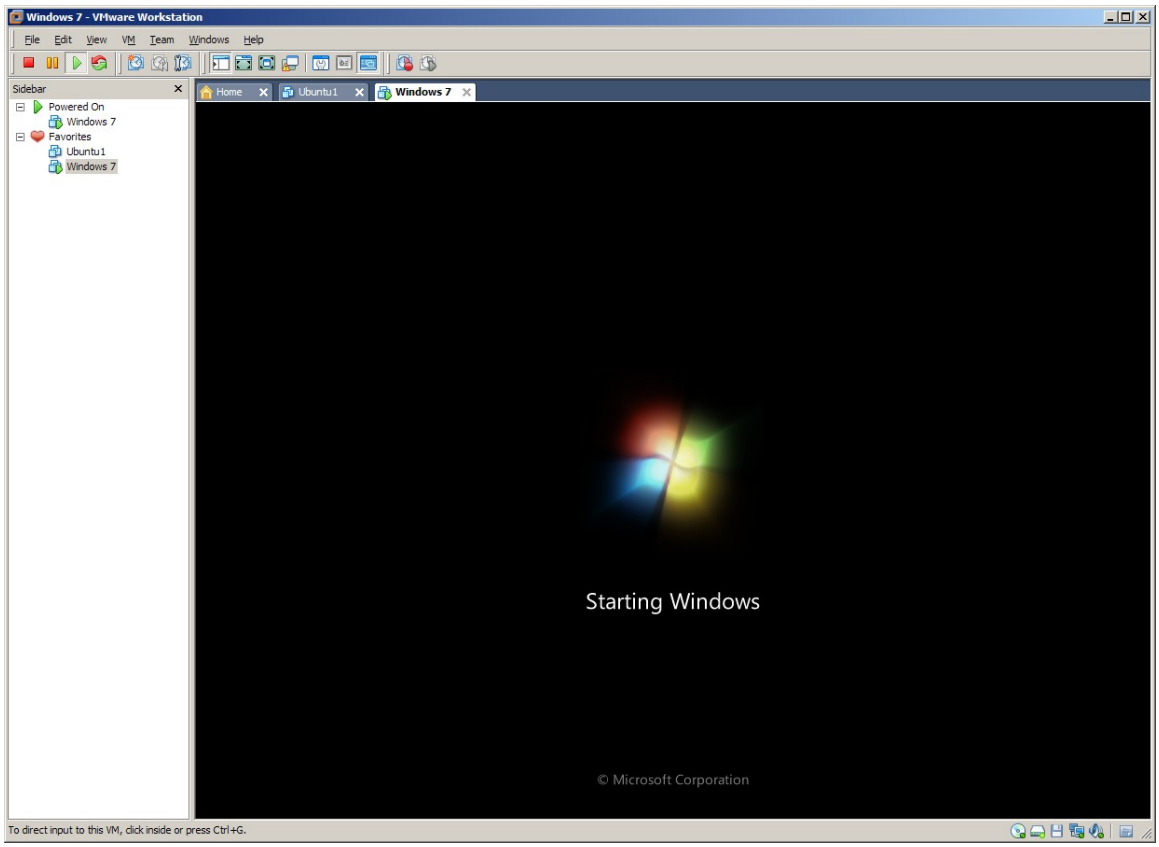
Add these lines to the file:

```
debugStub.listen.guest32 = "TRUE"
debugStub.hideBreakpoints= "TRUE"
```

Save the file.

In VMWare, click "Power on this virtual machine" or click the green Play button on the

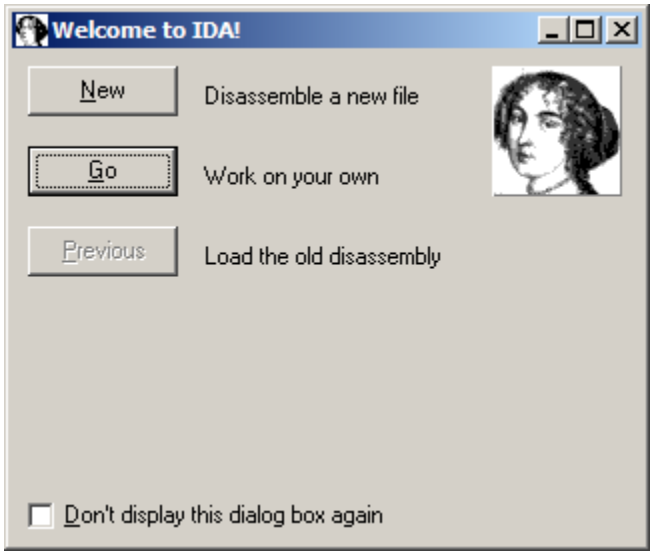
toolbar.



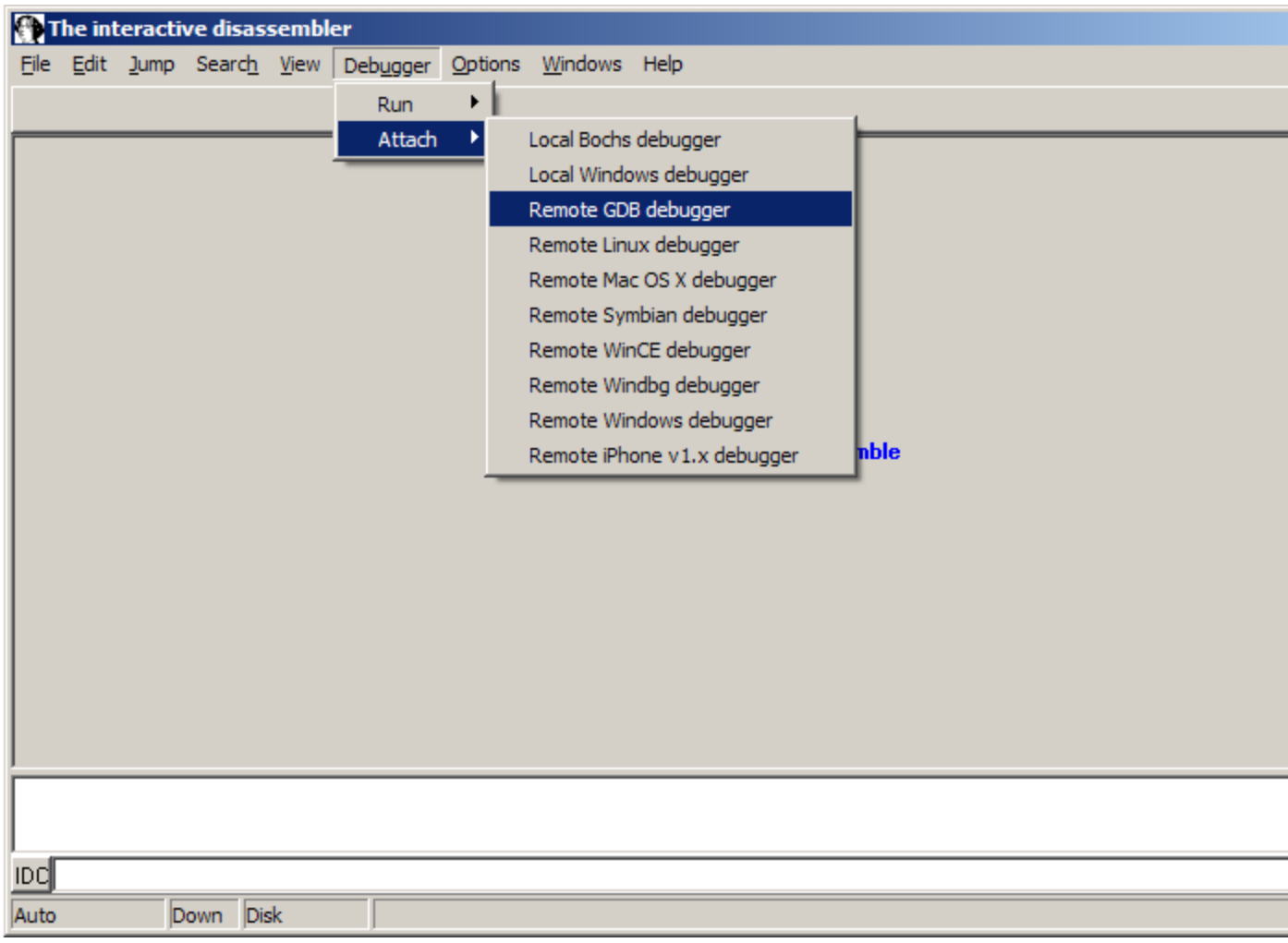
Wait until the VM boots.

Debugging in IDA

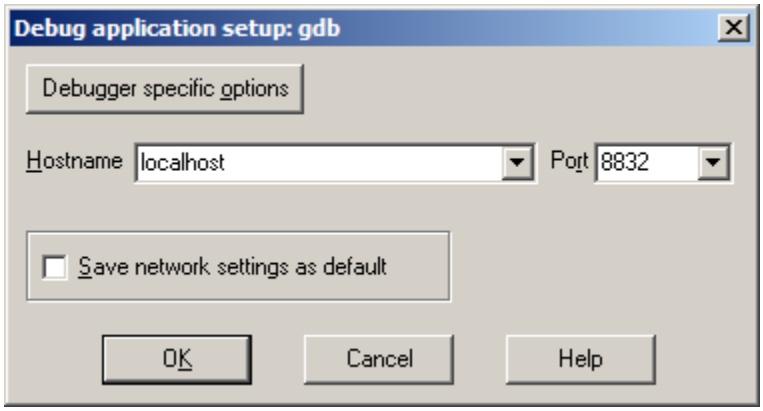
Start IDA.



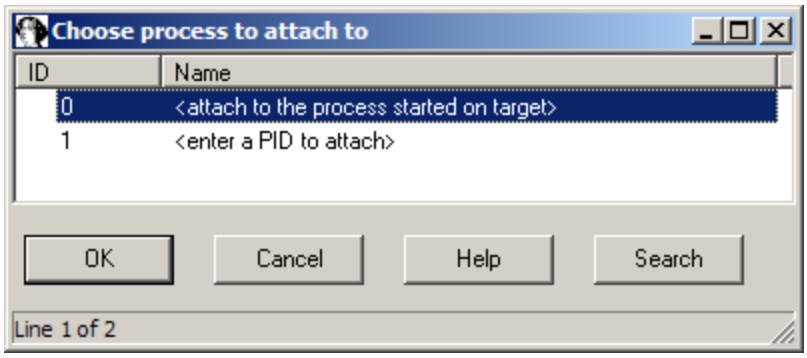
If you get the welcome dialog, choose "Go".



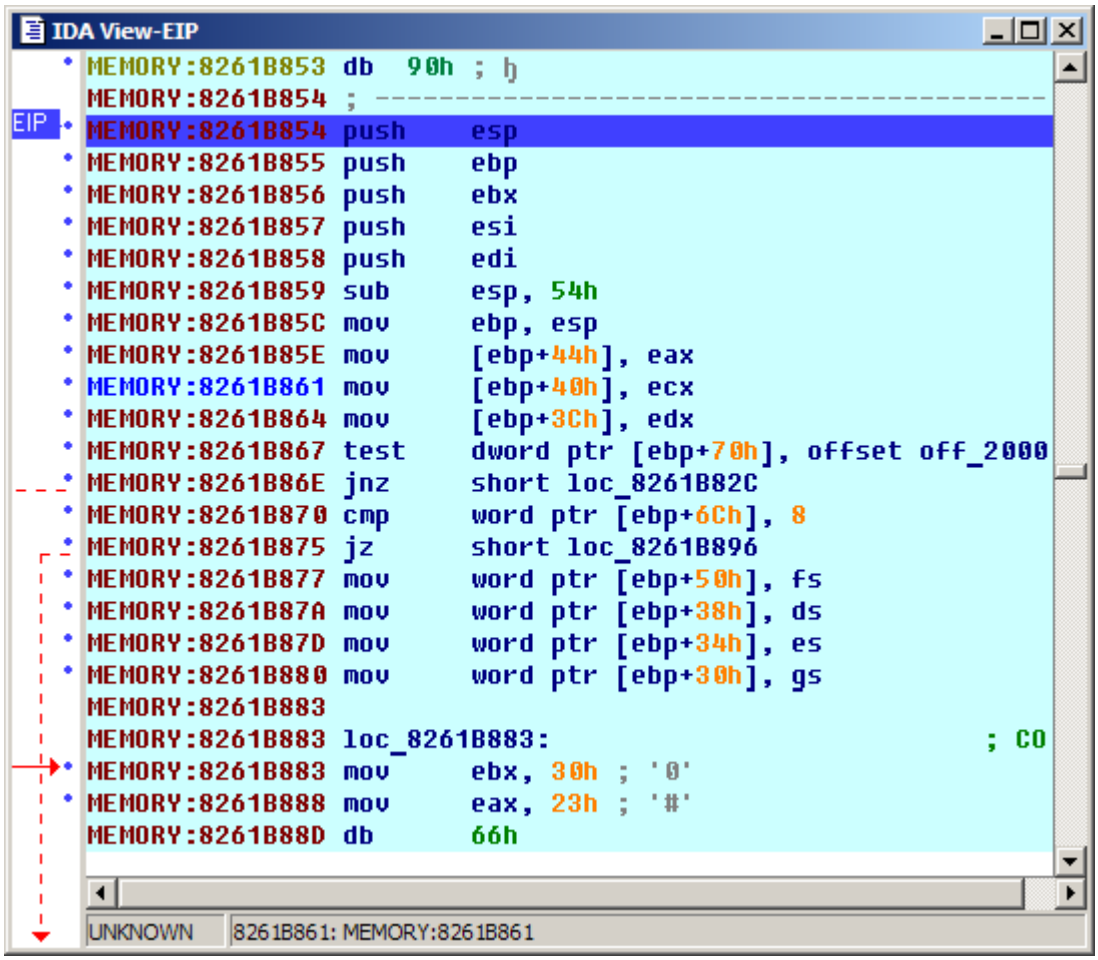
Choose Debugger | Attach | Remote GDB debugger.



Enter "localhost" for hostname and 8832 for the port number.



Choose <attach to the process started on target> and click OK.



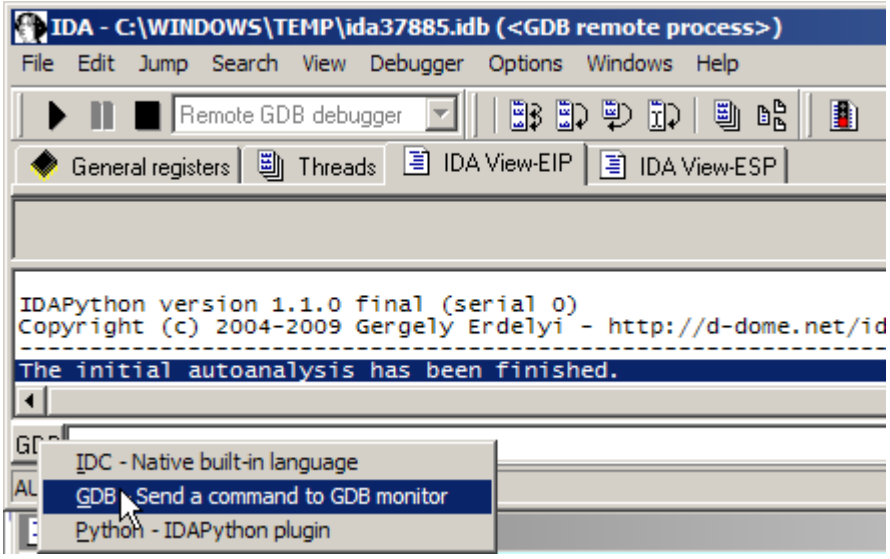
The execution should stop somewhere in the kernel (address above 0x80000000). You can step through the code, but it's not very convenient without any names. Let's try to add some symbols.

Identifying kernel base

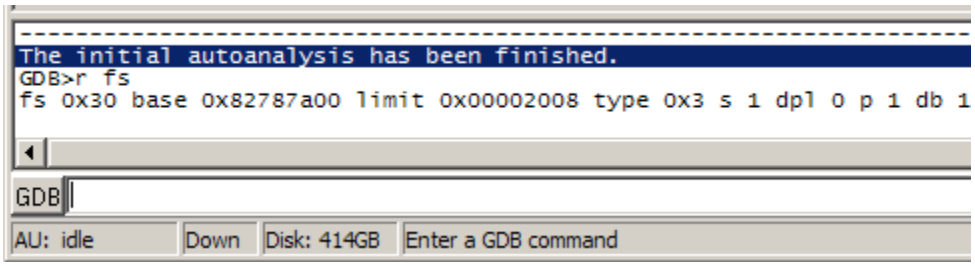
To load the kernel symbols from PDB file, we first need to know at which address it is loaded. There are many ways to find it, but we will use the `KernBase` pointer in the

DBGKD_GET_VERSION64 structure, which, in turn, is referenced from the KPCR structure pointed to by the fs register.

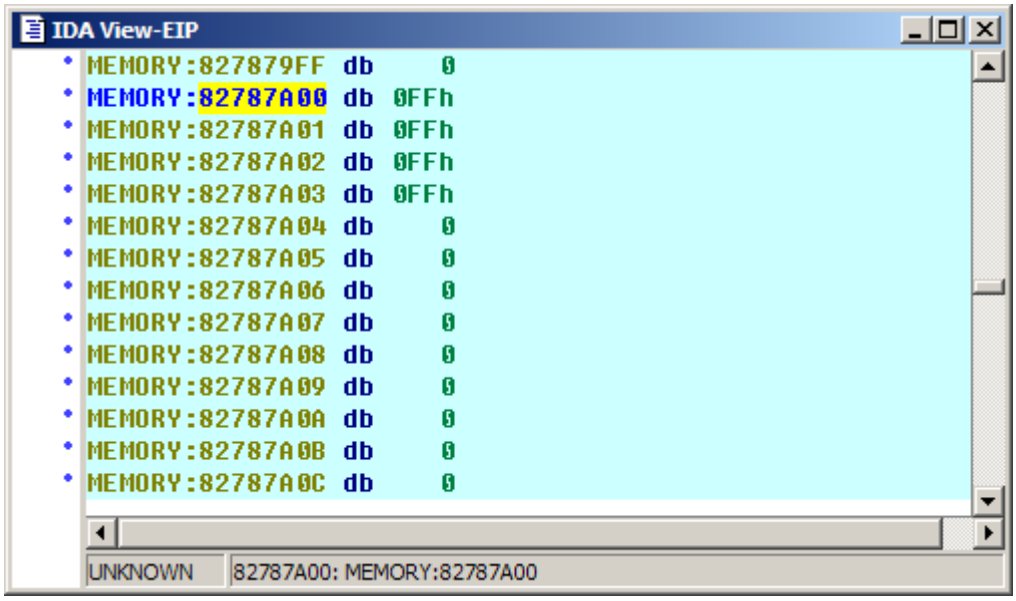
To retrieve the base of the segment pointed to by fs, we can use the VMWare's debug monitor "r" command. Make sure you switch the commandline interpreter to "GDB":



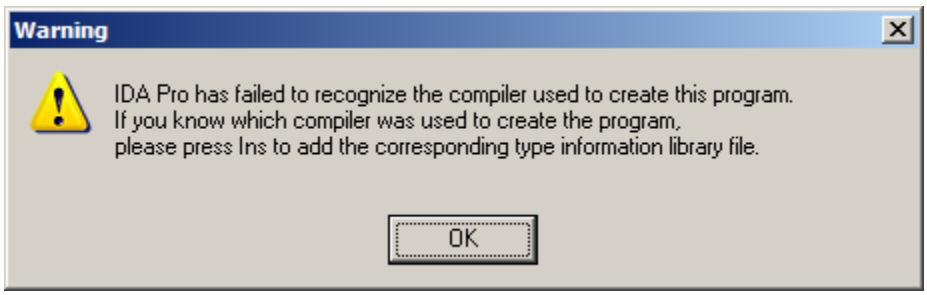
Then type "r fs" in the commandline (hint: use "." to switch to commandline and Esc to switch back to disassembly):



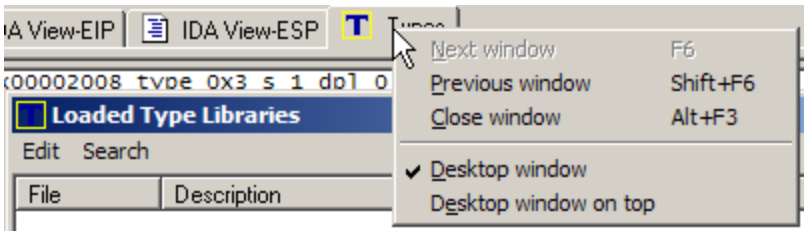
The address mentioned after "base" will be the address of KPCR.
Navigate to the address in IDA (Jump | Jump to addres... or just "g"). You will see the start of the KPCR structure.



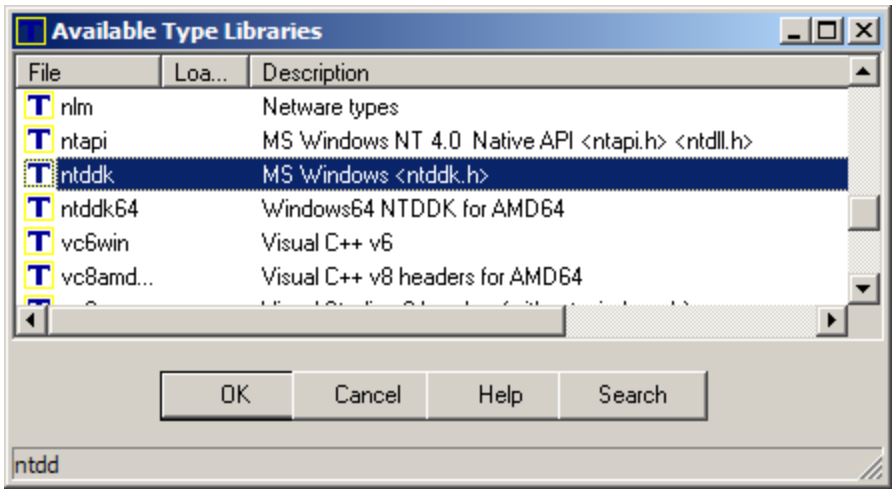
We could just follow some well-known offsets, but for convenience we will make use of the structure definitions bundled with IDA.
Choose View | Open subviews | Type libraries or press Shift-F11. You might get this warning, which is normal in this situation:



Press OK to close it. If you cannot see the Loaded Type Libraries window, right-click the Types tab on the main IDA window and make sure "Desktop window" item is checked:

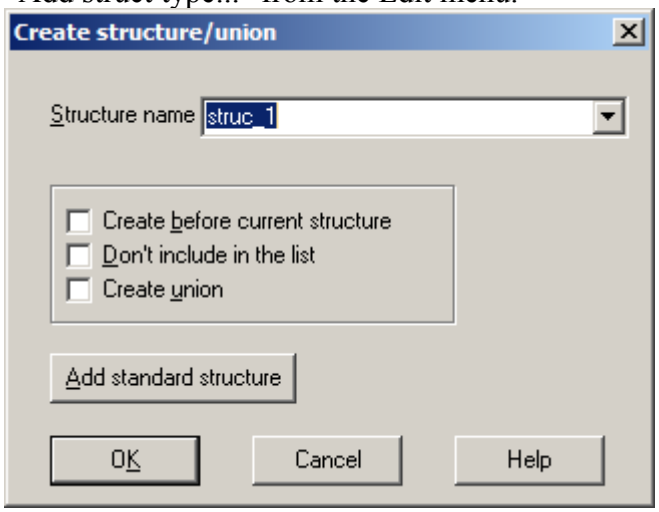


In the Loaded Type Libraries window choose "Load type library..." from the context menu or press Ins.

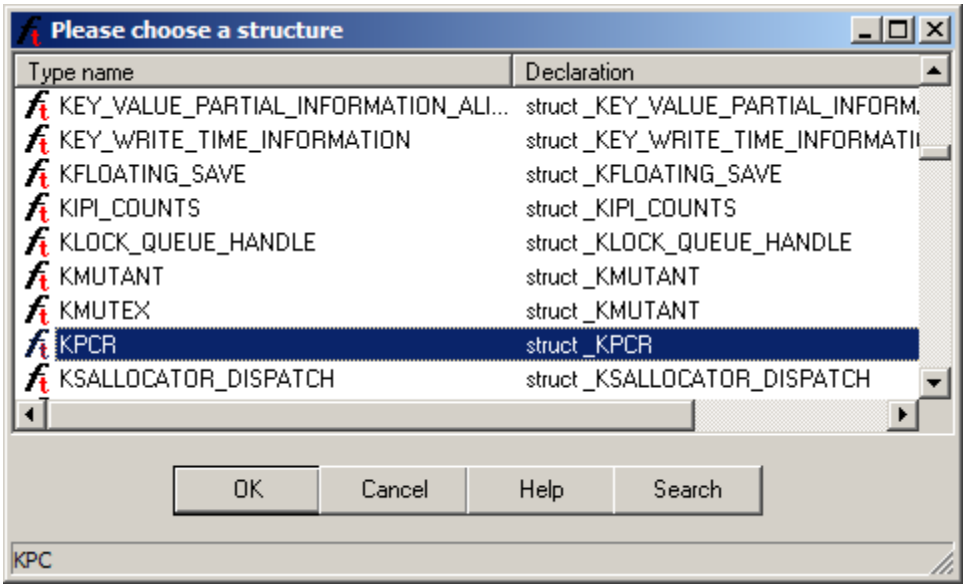


In the list, choose file named "ntddk" (hint: you can start typing the name to quickly navigate to it) and click OK. This will make available various structures from the Windows Device Driver Kit.

Now open the structures window (View | Open subviews | Structures or Shift-F9). There, press Ins or choose "Add struct type..." from the Edit menu.

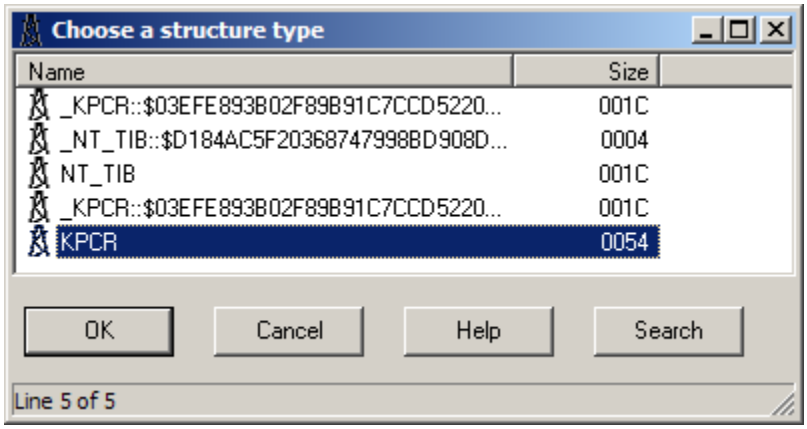


Click "Add standard structure". The list of predefined structures we loaded will appear.

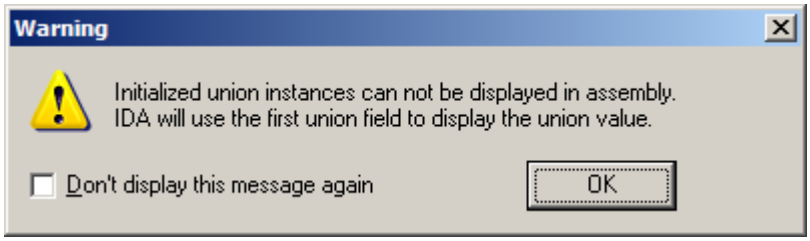


In the list, choose structure named KPCR (again, you can start typing its name). Click OK.

Now let's go back to the disassembly and apply the structure. Make sure you're at the address (fs base) we found earlier, and choose Edit | Structs | Struct var... or just press Alt-Q.



Choose the KPCR structure and click OK. You might get this warning, which can be ignored:



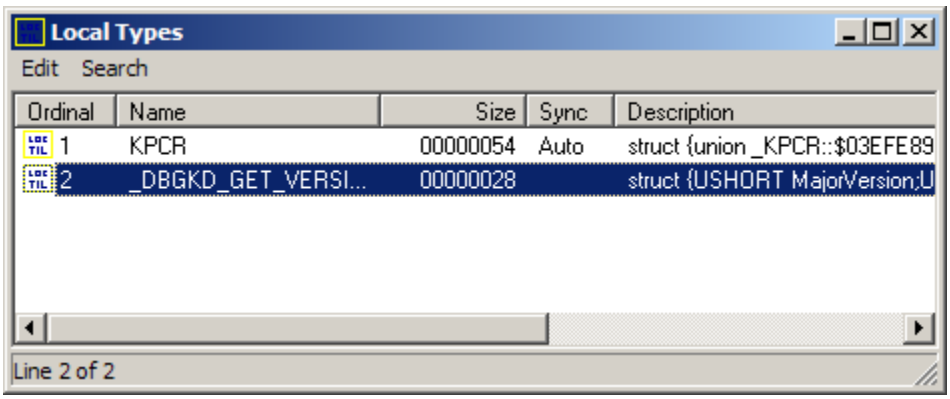
The dissassembly will display structure fields nicely represented and commented.

```
IDA View-EIP
MORY:82787A00 stru_82787A00 dd 0FFFFFFFFh                                  ; a
MORY:82787A00                                                  ; DATA XREF: MEMO
MORY:82787A00 dd 0                                             ; anonymous_0.NtT
MORY:82787A00 dd 0                                             ; anonymous_0.NtT
MORY:82787A00 dd offset unk_801DA000                           ; anonymous_0.NtT
MORY:82787A00 dd offset unk_2FDDE6                             ; anonymous_0.NtT
MORY:82787A00 dd offset unk_1                                  ; anonymous_0.NtT
MORY:82787A00 dd 0                                             ; anonymous_0.NtT
MORY:82787A00 dd offset stru_82787A00                          ; SelfPcr
MORY:82787A00 dd offset off_82787B20                           ; Prcb
MORY:82787A00 db 2                                             ; Irql
MORY:82787A00 db 3 dup(0)
MORY:82787A00 dd 0                                             ; IRR
MORY:82787A00 dd 0                                             ; IrrActive
MORY:82787A00 dd 0FFFFFFFFh                                    ; IDR
MORY:82787A00 dd offset off_82786A88                           ; KdVersionBlock
MORY:82787A00 dd offset off_83540400                           ; IDT
MORY:82787A00 dd offset unk_83540000                           ; GDT
MORY:82787A00 dd offset unk_801DA000                           ; TSS
UNKNOWN   82787A00: MEMORY:stru_82787A00
```

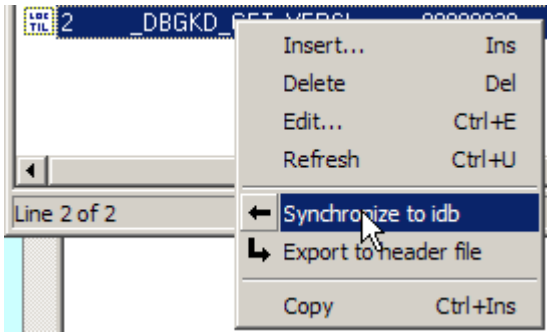
Find the field commented `KdVersionBlock` and double-click or press Enter to follow it. Name the location `KdVersionBlock` so that it can be easily found later.
Now we need to apply a structure `DBGKD_GET_VERSION64`, but it is not available in the type libraries shipped with IDA. Its definition is available in the file `wdbgexts.h` shipped with Debugging Tools for Windows or can be found on the Internet. We could create the structure manually from its definition, but it's much easier to use IDA's built-in C parser.
Go to View | Open subviews | Local types or press Shift-F1. Press Ins or choose "Insert..." from the context menu.

```
Please enter text
Please enter a new type declaration
    // debugger extensions.  This is the address of t
    //
    // There will always be an entry for the debugger
    //

    ULONG64 DebuggerDataList;

} DBGKD_GET_VERSION64, *PDBGKD_GET_VERSION64;

OK    Cancel    Help
```

Paste the definition of the `DBGKD_GET_VERSION64` structrure from `wdbgexts.h` file or Internet. Click OK.



The structure will appear in the Local Types list, but it will not be available yet in the Structures list or in Struct var... command.



Right-click it and choose "Synchronize to idb". Click Yes in the confirmation notice and OK in the info messagebox. Now we can apply the structure.
Go to the `KdVersionBlock` location we noted before and apply the new structure (using Edit | Structs | Struct var... or Alt-Q). The structure will initially appear collapsed, press Numpad + to uncollapse it (or choose Unhide from the View menu).

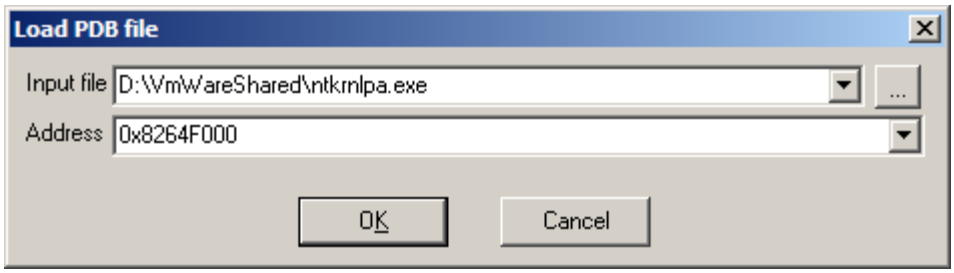
```
IDA View-EIP
MEMORY:82786A83 db    0
MEMORY:82786A84 db   92h ; '
MEMORY:82786A85 db    0
MEMORY:82786A86 db    0
MEMORY:82786A87 db    0
MEMORY:82786A88 KdVersionBlock dw 0Fh                          ; Majo
MEMORY:82786A88                                  ; DATA XREF: MEMORY:s
MEMORY:82786A88 dw 1B58h                         ; MinorVersion
MEMORY:82786A88 db 6                             ; ProtocolVersion
MEMORY:82786A88 db 0                             ; KdSecondaryVersion
MEMORY:82786A88 dw 3                             ; Flags
MEMORY:82786A88 dw 14Ch                          ; MachineType
MEMORY:82786A88 db 0Ch                           ; MaxPacketType
MEMORY:82786A88 db 3                             ; MaxStateChange
MEMORY:82786A88 db 2Fh                           ; MaxManipulate
MEMORY:82786A88 db 0                             ; Simulation
MEMORY:82786A88 dw 0                             ; Unused
MEMORY:82786A88 dq 0FFFFFFFF8264F000h            ; KernBase
MEMORY:82786A88 dq 0FFFFFFFF827A6830h            ; PsLoadedModuleList
MEMORY:82786A88 dq 0FFFFFFFF829C4FECh            ; DebuggerDataList
MEMORY:82786AB0 db 0ECh ; м
UNKNOWN   82786A88: MEMORY:KdVersionBlock
```

Now everything is nicely represented, and we can follow the `KernBase` pointer (the Fs at the beginning of the address are caused by sign extension and can be ignored here; double-click or Enter key still work). After following the pointer, you can see the familiar MZ signature.

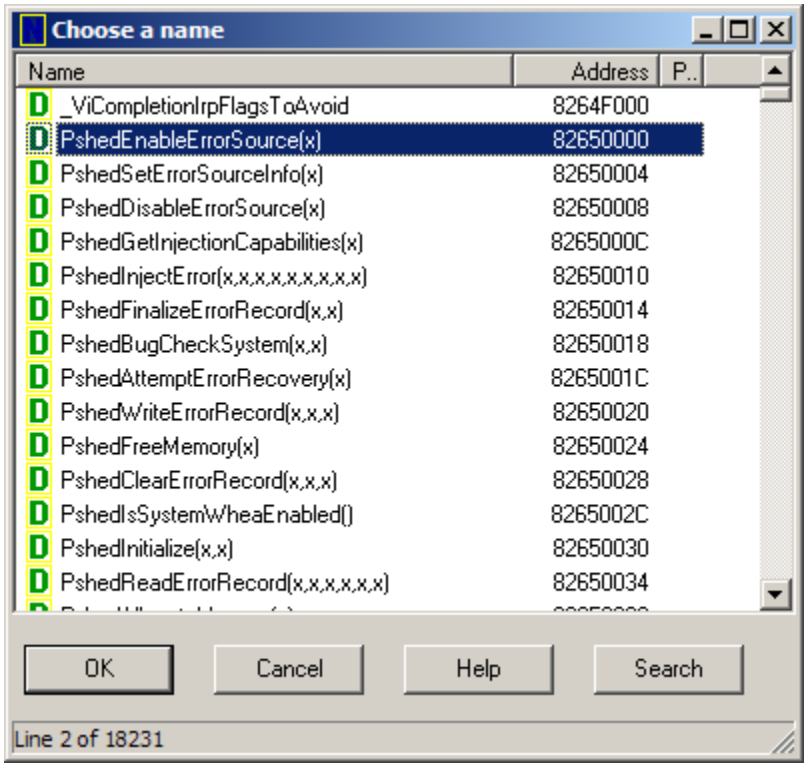
```
IDA View-EIP
MEMORY:8264EFFE db    ? ;
MEMORY:8264EFFF db    ? ;
MEMORY:8264F000 unk_8264F000 db  4Dh ; M
MEMORY:8264F001 db  5Ah ; Z
MEMORY:8264F002 db  90h ; ‡
MEMORY:8264F003 db    0
MEMORY:8264F004 db    3
MEMORY:8264F005 db    0
UNKNOWN   8264F000: MEMORY:unk_8264F000
```

Loading symbols

To load the symbols, we will make use of the new feature IDA 5.4. Go to File | Load File | PDB file...
Enter path to the kernel .exe you copied from the VM and the kernel base address we found.



Press OK and IDA's PDB loader plugin will retrieve the PDB file for the .exe from Microsoft's symbol store and apply its symbols to the disassembly.



This should make your work much easier.

Happy debugging!