

# Debugging Linux kernel under VMWare using IDA's GDB debugger

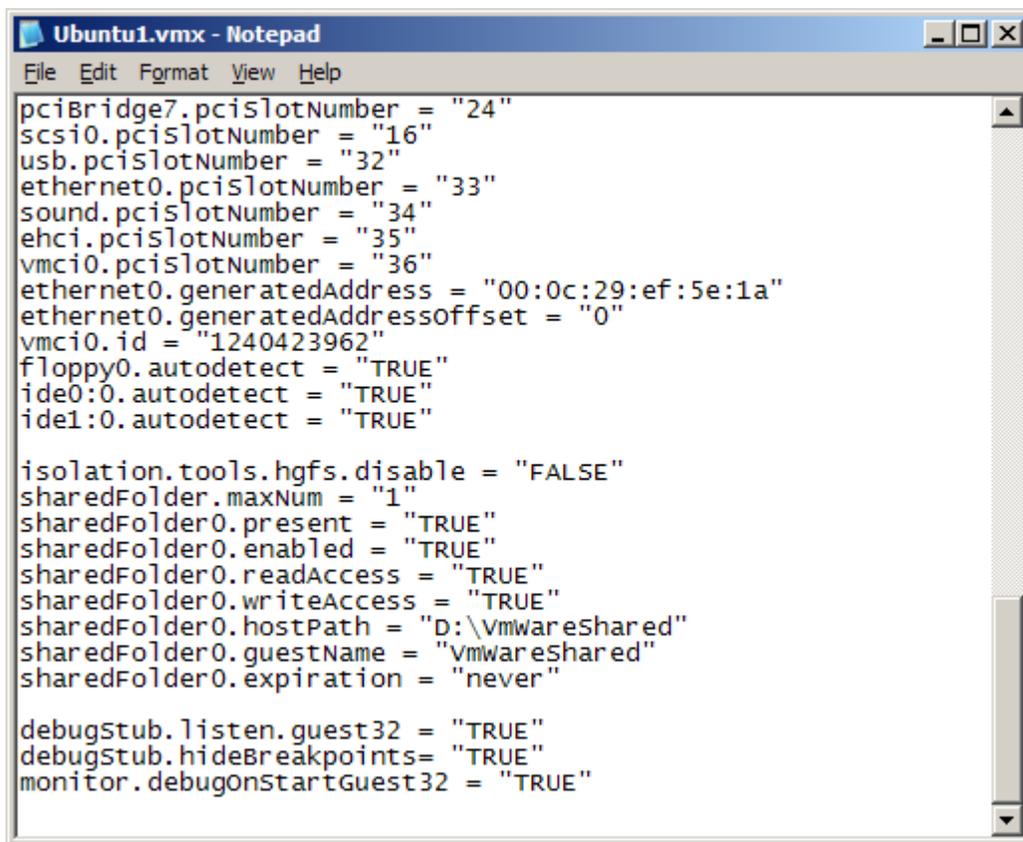
Copyright 2009 Hex-Rays SA

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 Linux kernel.

## Debugging a Linux kernel

Let's assume that you already have a VM with Linux installed. Before starting the debugging, we will copy symbols for the kernel for easier navigation later. Copy either `/proc/kallsyms` or `/boot/Sytem.map*` file from the VM to host.

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



```
pciBridge7.pciSlotNumber = "24"
scsi0.pciSlotNumber = "16"
usb.pciSlotNumber = "32"
ethernet0.pciSlotNumber = "33"
sound.pciSlotNumber = "34"
ehci.pciSlotNumber = "35"
vmci0.pciSlotNumber = "36"
ethernet0.generatedAddress = "00:0c:29:ef:5e:1a"
ethernet0.generatedAddressOffset = "0"
vmci0.id = "1240423962"
floppy0.autodetect = "TRUE"
ide0:0.autodetect = "TRUE"
ide1:0.autodetect = "TRUE"

isolation.tools.hgfs.disable = "FALSE"
sharedFolder.maxNum = "1"
sharedFolder0.present = "TRUE"
sharedFolder0.enabled = "TRUE"
sharedFolder0.readAccess = "TRUE"
sharedFolder0.writeAccess = "TRUE"
sharedFolder0.hostPath = "D:\VmwareShared"
sharedFolder0.guestName = "VmwareShared"
sharedFolder0.expiration = "never"

debugStub.listen.guest32 = "TRUE"
debugStub.hideBreakpoints= "TRUE"
monitor.debugOnStartGuest32 = "TRUE"
```

Add these lines to the file:

```
debugStub.listen.guest32 = "TRUE"
debugStub.hideBreakpoints= "TRUE"
monitor.debugOnStartGuest32 = "TRUE"
```

Save the file.

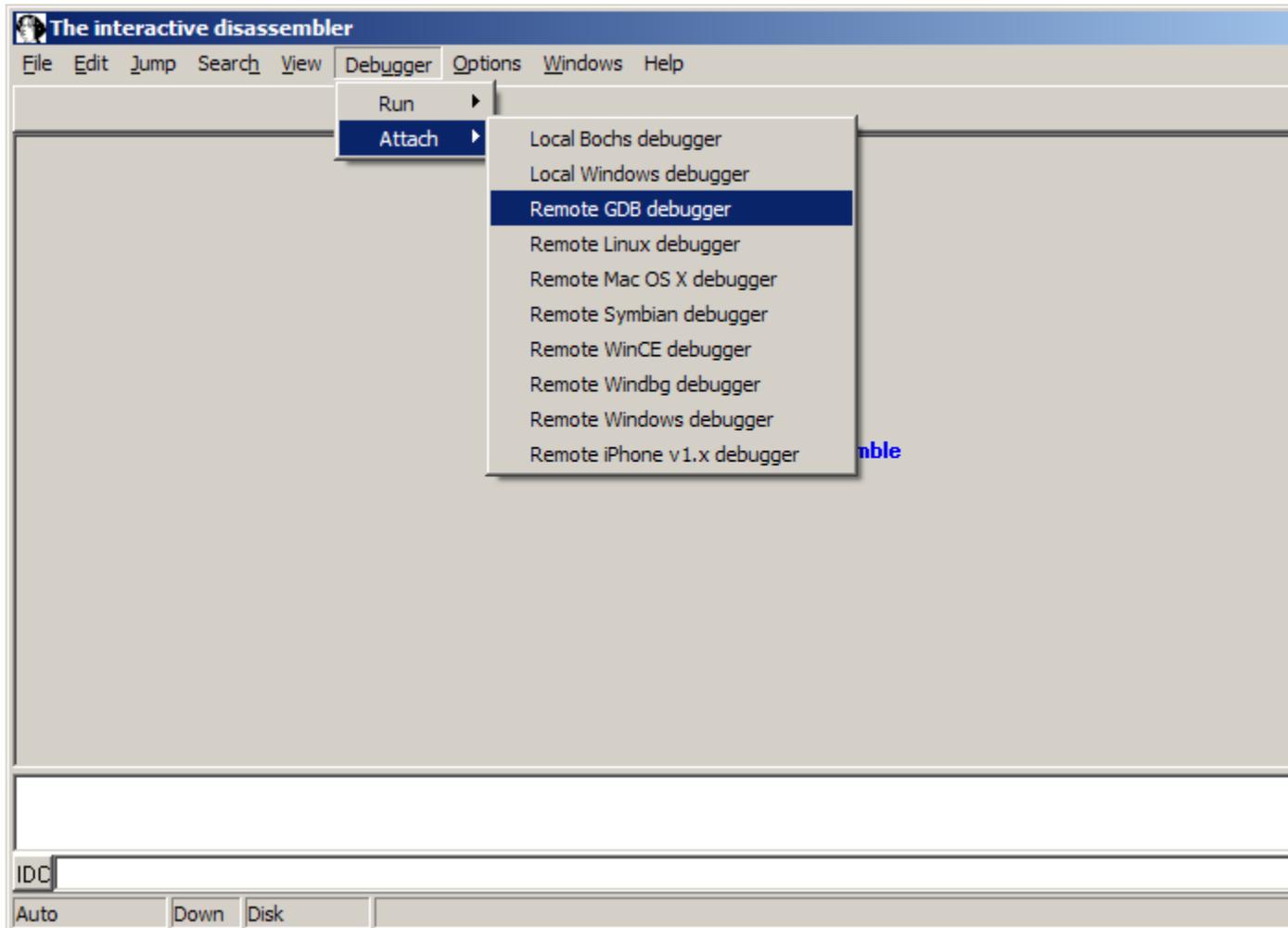
In VMWare, click "Power on this virtual machine" or click the green Play button on the toolbar.

A black screen is displayed since VMWare is waiting for a debugger to connect.

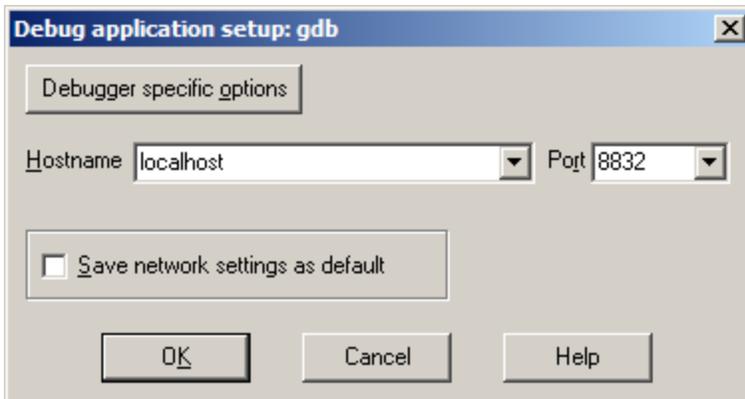
Start IDA.



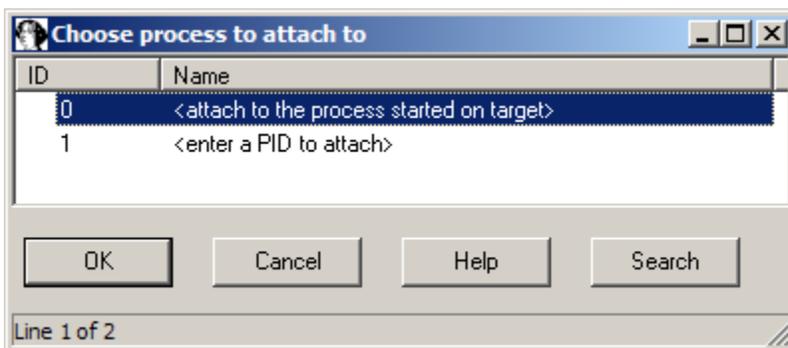
If you get the welcome dialog, choose "Go".



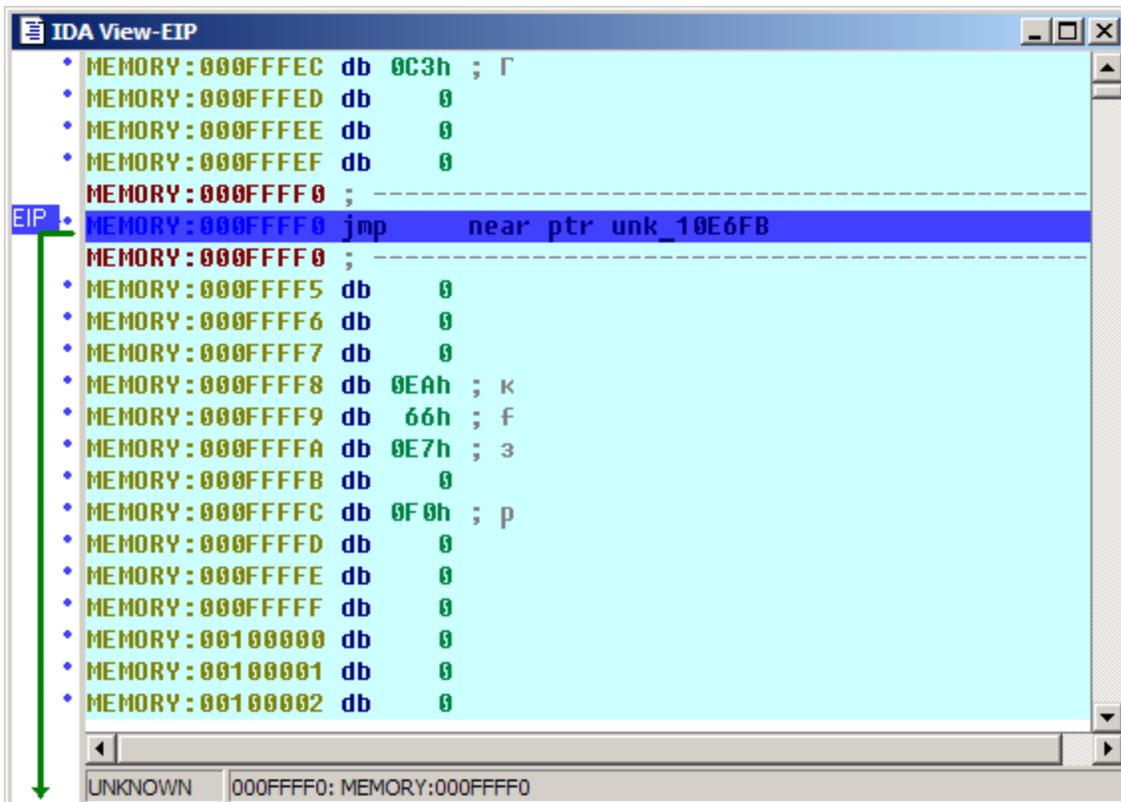
Choose Debugger | Attach | Remote GDB debugger.



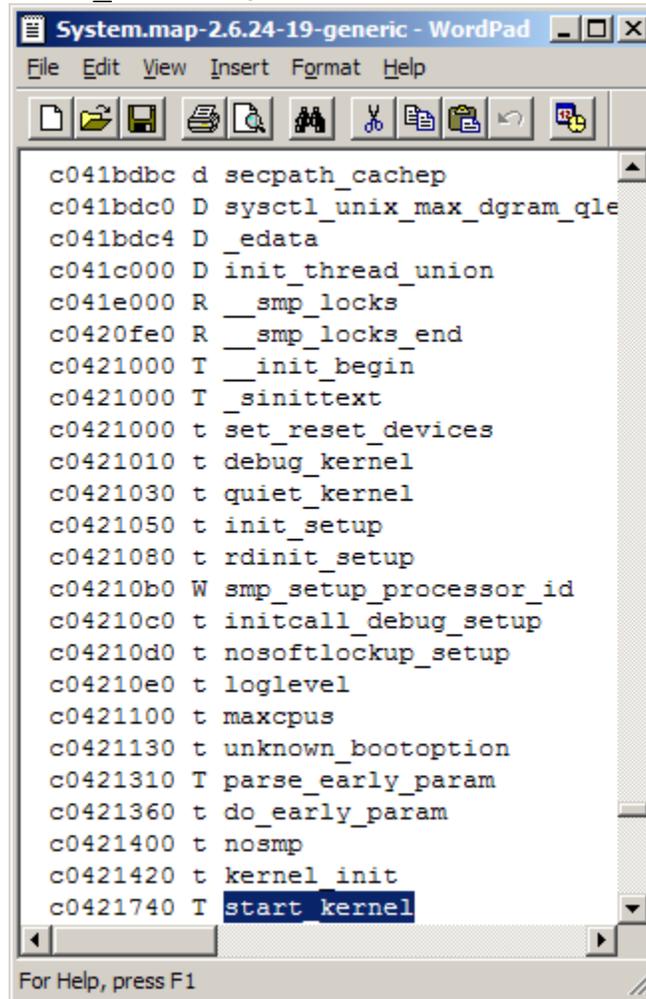
Enter "localhost" for hostname and 8832 for the port number.



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

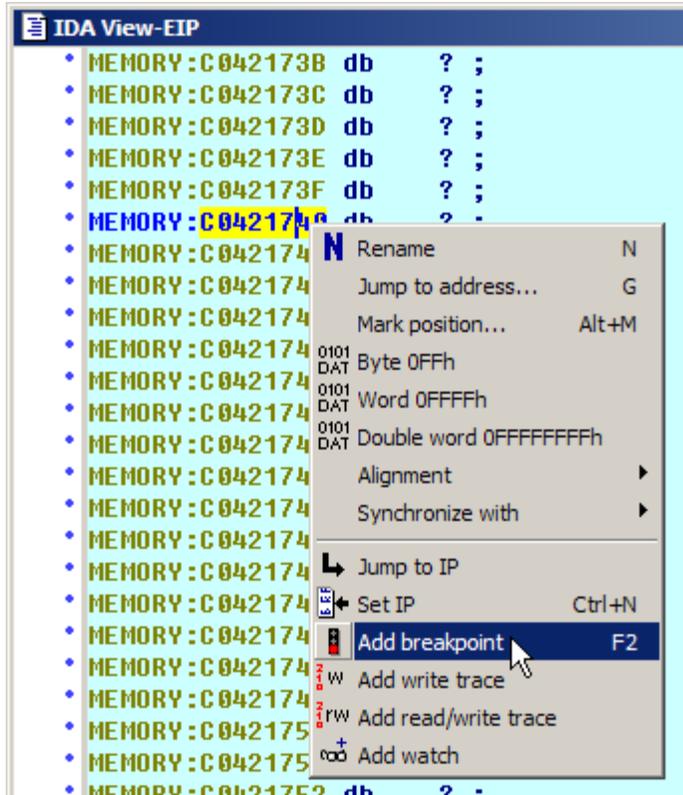


We land in the BIOS, but since we're not interested in debugging it, we can skip directly to the kernel. Inspect the kallsyms or System.map file you downloaded from the guest and search for the `start_kernel` symbol:

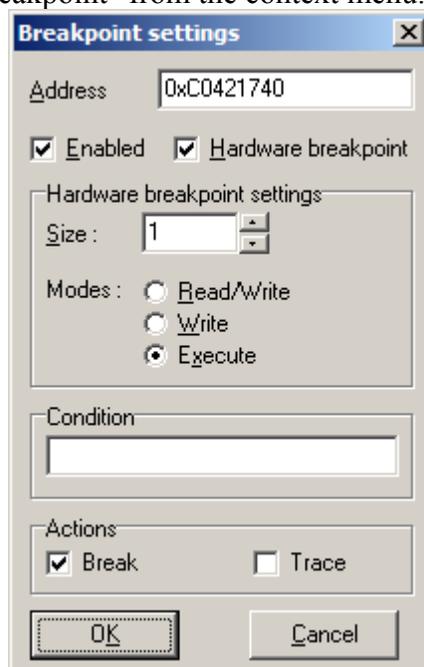


```
System.map-2.6.24-19-generic - WordPad
File Edit View Insert Format Help
c041bdbc d secpath_cachep
c041bdc0 D sysctl_unix_max_dgram_que
c041bdc4 D _edata
c041c000 D init_thread_union
c041e000 R __smp_locks
c0420fe0 R __smp_locks_end
c0421000 T __init_begin
c0421000 T __sinittext
c0421000 t set_reset_devices
c0421010 t debug_kernel
c0421030 t quiet_kernel
c0421050 t init_setup
c0421080 t rdinit_setup
c04210b0 W smp_setup_processor_id
c04210c0 t initcall_debug_setup
c04210d0 t nosoftlockup_setup
c04210e0 t loglevel
c0421100 t maxcpus
c0421130 t unknown_bootoption
c0421310 T parse_early_param
c0421360 t do_early_param
c0421400 t nosmp
c0421420 t kernel_init
c0421740 T start_kernel
For Help, press F1
```

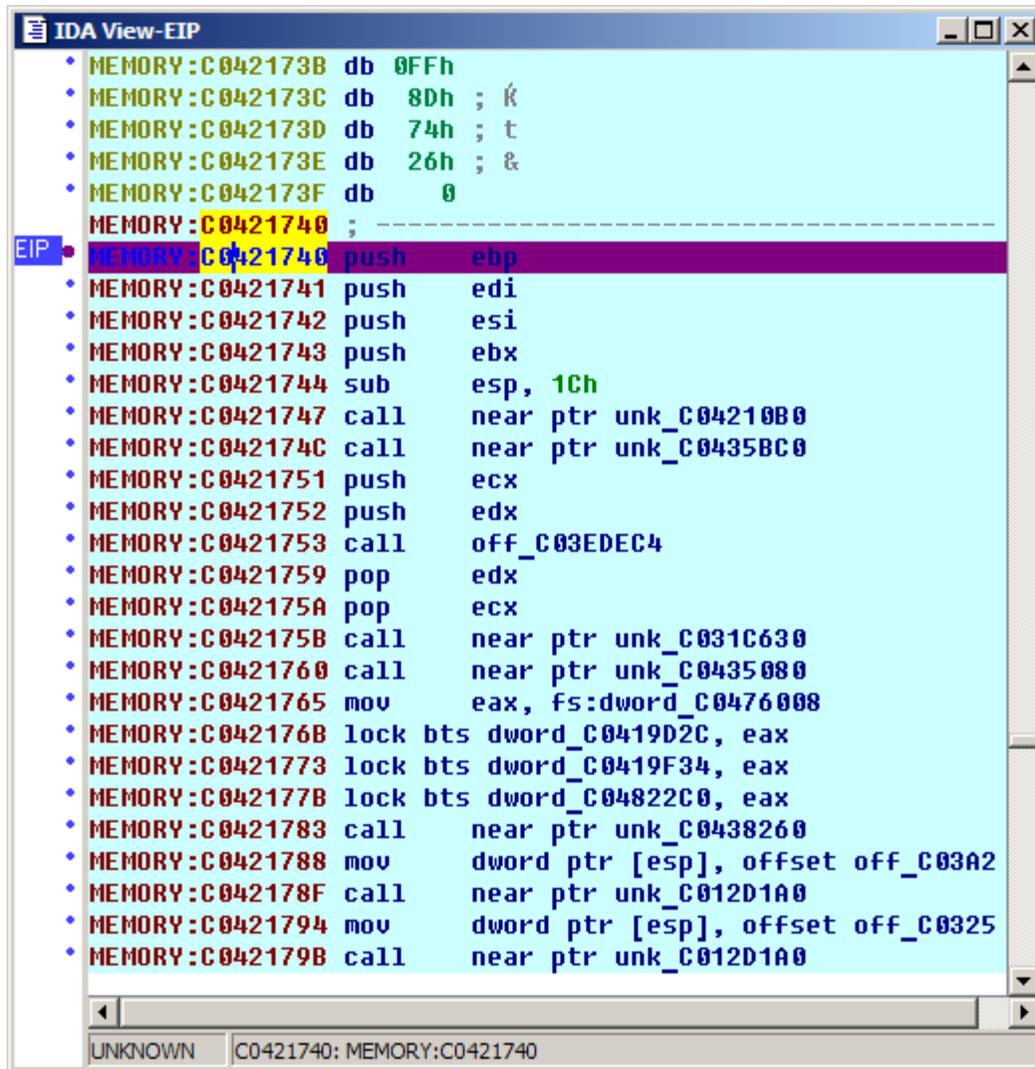
Copy the address, and navigate to it in IDA (Jump | Jump to address... or just "g").



Press F2 or choose "Add breakpoint" from the context menu.



Check "Hardware breakpoint" and select "Execute" in "Modes". Click OK. Press F9. You will see loading messages and then the execution will stop at the breakpoint.



## Adding symbols

Symbols are very useful during debugging, and we can use the kallsyms or System.map file to add them to IDA. Go to File | Python command... and paste the following short script (don't forget to edit the file path):

```

ksyms = open(r"D:\kallsyms") # path to the kallsyms/map file
for line in ksyms:
    if line[9]=='A': continue # skip absolute symbols
    addr = int(line[:8], 16)
    name = line[11:-1]
    if name[-1]==' ': continue # skip module symbols
    idaapi.set_debug_name(addr, name)
    MakeNameEx(addr, name, SN_NOWARN)
    Message("%08X: %s\n"%(addr, name))

```

Click OK and wait a bit until it finishes. After that you should see the symbols in the

disassembly and name list:

The screenshot shows the IDA View-EIP window with the following disassembly:

```
MEMORY:C042173B db 0FFh
MEMORY:C042173C db 8Dh ; k
MEMORY:C042173D db 74h ; t
MEMORY:C042173E db 26h ; &
MEMORY:C042173F db 0
MEMORY:C0421740 ; -----
MEMORY:C0421740
MEMORY:C0421740
MEMORY:C0421740 start_kernel:
EIP MEMORY:C0421740 push ebp
MEMORY:C0421741 push edi
MEMORY:C0421742 push esi
MEMORY:C0421743 push ebx
MEMORY:C0421744 sub esp, 1Ch
MEMORY:C0421747 call near ptr smp_setup_processor_id
MEMORY:C042174C call near ptr cgroup_init_early
MEMORY:C0421751 push ecx
MEMORY:C0421752 push edx
```

The 'Choose a name' dialog box is open, showing a list of names and addresses:

Name	Address	P..
nosoftlockup_setup	C04210D0	
loglevel	C04210E0	
unknown_bootoption	C0421130	
parse_early_param	C0421310	
do_early_param	C0421360	
nosmp	C0421400	
kernel_init	C0421420	
<b>start_kernel</b>	<b>C0421740</b>	
readonly	C0421AF0	
readwrite	C0421B10	
rootwait_setup	C0421B30	
root_data_setup	C0421B50	
fs_names_setup	C0421B60	

The dialog box has buttons for OK, Cancel, Help, and Search. The status bar at the bottom indicates 'Line 24580 of 27466'.

Happy debugging!