Replay Debugging on Linux

VMware Workstation 7.0

Introduction

This technical note constitutes a quick-start guide to software development and debugging on Linux using VMware Workstation and Replay Debugging. Replay Debugging uses the Record and Replay feature of Workstation to make a debugging session deterministic, repeatable, and reversible.

What Is Record and Replay?

Record and Replay allows you to run a program in a virtual machine under Workstation, record the entire execution of the program, including all of its input and output, and later repeat (or "replay") the entire execution, or any part of it, as often as you want, with every instruction, and every input or output, precisely and deterministically identical to the original execution.

What Is Replay Debugging?

With Replay Debugging, you run a debugger (in this case, `gdb`) on your host computer, while you replay a previously recorded program that you want to debug in your "guest" machine (the virtual machine running within Workstation). The `gdb` debugger runs in your host Linux environment and connects to the program running in a pre-recorded, virtual Linux environment with Workstation. This allows you to stop the program, examine its variables, step, set breakpoints, and continue the program as if you were debugging a live process on your own or another Linux host.

Because the program being debugged runs in a recording, everything happens exactly the way it happened during the original recorded execution. Every event, including non-deterministic events such as inputs and task switches, occur in the same order, at the same point in the recorded timeline. When `gdb` stops the program and allows you to see what is going on, it stops the entire virtual machine, so that all other recorded processes are stopped. When you continue the program you are debugging, the entire virtual machine resumes at the same time, synchronously, without in any way "noticing" that it has been stopped. You can go back to the beginning and repeat your debugging session as often as you like, with full confidence that each repetition will execute identically.

What Is Reverse Debugging?

Because the program being debugged runs in a recording, you can back it up (rewind it) at any time during the replay, taking the execution back to the beginning or to an earlier point in the recording. When you are debugging a recording with `gdb`, you are effectively able to make the program run backward.

Anyone who has used a debugger has probably had the experience of suddenly realizing that you have accidentally gone too far – the event that you were looking for has passed, and you missed seeing it. With reverse debugging, instead of starting the program over from the beginning and repeating your entire (possibly lengthy) debugging session, you can set a breakpoint at an earlier point in the program, and "reverse-continue" so the program backs up and undoes itself to that earlier point. This feature allows you to proceed from that point forward again.

How to Use Replay Debugging

You need four components for Replay Debugging: VMware Workstation (32-bit only), `vmware-gdb-proxy` installed with Workstation, a 32-bit x86 Linux guest operating system, and `gdb` version 6.8 or later (version 7.0 or later for Reverse Debugging). The debugging process involves three steps:

1 Record the program using Workstation.

2 Start replaying the recorded program using `vmware-gdb-proxy`.

3 Attach to the recorded program and debug it using `gdb`.

NOTE Replay Debugging on Linux is experimental in the Workstation 7 release.

Making a Recording with Workstation

Before you make your first recording for replay debugging, you must take several preparatory steps.

You must create and install your virtual machine (VM). This document does not describe how to create a virtual machine – see the VMware Workstation documentation for instructions.

After you install the VM, you must add a few lines of customization to its VMX file. This file has a `.vmx` suffix and is located in the directory where you installed the VM on your host computer. Add the customization lines **before making your first recording** for replay debugging.

The first customization line, which you can add at the end of the VMX file, is `debugStub.linuxOffsets`. You must determine which Linux kernel distribution is running in your virtual machine, and select the proper line from Table 1, "Linux Kernel Offsets," on page 8. Add the `debugStub.linuxOffsets` values, exactly as shown in the table, to your VMX file. If you are not sure which Linux kernel is running in your virtual machine, start the virtual machine, open a terminal, and type the command `uname -a` at the shell prompt.

If you do not find the corresponding version of your Linux guest operating system listed in Table 1, see "Determining Kernel Offsets" on page 14 for some example code can derive the `debugStub.linuxOffsets` values for your Linux guest operating system.

The second customization line is `debugStub.stopAddress`. To get the value of `debugStub.stopAddress`, start your virtual machine, open a terminal, and type the following command:

```
host$ cat /proc/kallsyms | grep exit_mmap
```

Take the hexadecimal number that results and use it as the value of `debugStub.stopAddress`.

You can copy the third and fourth customization lines exactly as follows:

```
debugStub.listen.guest32 = “TRUE”
debugStub.listen.guest32.remote = “TRUE”
```

So you should end up with four new lines at the end of your VMX file, looking something like the following (substituting your own values):

```
debugStub.linuxOffsets = “0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10”
debugStub.stopAddress = "0xc0425eb1"
debugStub.listen.guest32 = "TRUE"
debugStub.listen.guest32.remote = "TRUE"
```

After you have added these values to your VMX file, if your virtual machine is running, you must stop it and start it again for these values to take effect. Suspend and resume is sufficient – you do not have to power off or reboot the virtual machine.

Now you are ready to run the program you want to debug, and make a recording. Presumably this program is one that you compile yourself. To prepare it for debugging with `gdb`, you should compile with the `-g` option to include debugging symbols.

It is your choice whether to compile the program on your host machine or within the guest virtual machine – assuming that the host and guest operating systems and shared libraries are compatible. If you compile the program on the host machine, you must copy it in to the guest machine's file system (or use a shared folder, NFS mount, or equivalent) to make it visible to the guest machine.

When you are ready to run the program in the guest, first begin by pressing the **Record** button on the Workstation toolbar. You see a dialog box saying "Saving virtual machine state to snapshot `xxx`." After that you see a widget with buttons and a progress bar. Workstation is now making a recording of the guest machine's execution.

Start your program and run it as usual. When it is finished, go to the recording widget and press **Stop**. You are given an opportunity to name to your recording before you save it. Make a note of this name for the next step.

You are done with the first step, making the recording.

Replaying a Recording with vmware-gdb-proxy

To replay a recording, we must run a program called `vmware-gdb-proxy` on the host machine. The `vmware-gdb-proxy` provides a bridge between `gdb` (running in the host Linux environment) and the program to be debugged (running in the virtual machine guest Linux environment). The proxy also helps `gdb` to control Workstation and the virtual machine.

To enable `gdb` to debug the program, you must have a copy of both the source files and the compiled binary executable file available within the host machine's file system. If you did your compiling in the guest, you might need to copy these files, or use a shared folder or NFS mount.

The proxy `vmware-gdb-proxy` has a number of command-line options (see "Command Line Options for vmware-gdb-proxy" on page 6), but the simplest way to start your recording and prepare it for debugging is to give `vmware-gdb-proxy` these three options:

`-V` with the path to the VMX file for the virtual machine
`-R` with the name of the recording, and
`-F` with the path to the program executable (in the host file system).

For example, if your virtual machine is located in `/home/user/vms/Ubuntu904/Ubuntu904.vmx`, your recording is named `Recording1`, and your program executable is located in `builds/Ubuntu/myprog`, the command line would look like this:

```
vmware-gdb-proxy -V /home/user/vms/Ubuntu904/Ubuntu904.vmx -R Recording1 -F builds/Ubuntu/myprog
```

This should cause Workstation to open (if it is not already running), power on the virtual machine, start replaying the recording, and play it to the point where your program starts.

At that point, the virtual machine pauses, and `vmware-gdb-proxy` prints a message in its terminal:

```
waiting for commands from gdb...
```

You are now ready to start `gdb` and begin Replay Debugging.

Attaching to the Recorded Program with gdb

Replay debugging requires `gdb` version 6.8 or later. Reverse debugging requires `gdb` version 7.0 or later.

When `vmware-gdb-proxy` is ready for `gdb`, you can start `gdb` on your host machine with the binary executable file for the program that you want to debug. In the example above, assume that the binary file is located at `builds/Ubuntu/myprog`. Start `gdb` from the shell prompt:

```
host$ gdb builds/Ubuntu/myprog
```

When the `gdb` prompt appears, type the following `gdb` instruction:

```
(gdb) target remote localhost:7732
```

In this example, `7732` is the local socket ID that `vmware-gdb-proxy` uses by default. If you want to use a different socket ID, include the `-p` (port) option to `vmware-gdb-proxy` as described below. The debugger now does hand-shaking with `vmware-gdb-proxy`, and should shortly inform you that you are at the start address of your program (something like this):

```
0x080486c0 in _start ()
(gdb)
```

Your program is now at its first executable runtime instruction (note – before `main`), and ready to run. If you want to go to main, type `break main`, and then type `continue`.

From this point, you can set breakpoints, print variables, examine variables, display registers, step, next, and continue, much as if you were debugging a native process on your host. The one principal thing you cannot do is change anything! Because you are debugging a recording, and not a live process, you cannot change the values of memory or registers, and you cannot change the execution path that your program takes. Everything is deterministic, and therefore predetermined.

How to Use Reverse Debugging

If you have `gdb` version 7.0 or higher (or if you can download and build `gdb` from the `gdb` developers' CVS source tree), you have access to one more debugging feature – `reverse-continue`.

As you know, in the normal course of debugging, you can set some breakpoints, run for a while, stop at a breakpoint, step for a while, run some more, and so forth, all the time looking at your program's data to try and find a bug. It has probably happened to you occasionally that you type `continue` a few times in a row, and suddenly realize that your program has gone too far, and that you missed the event you were looking for.

But with Replay Debugging and `gdb` 7.0, if you encounter this situation, you can type `reverse-continue`, and `gdb` asks Workstation to run the program backwards until it hits the previous breakpoint or watchpoint. In most typical debugging scenarios, this can be much easier and faster than starting your entire debugging session over from the beginning and repeating all of your debugging steps except the last one.

With both continue and reverse-continue, you can now go directly from any point in the program's execution to any other point, forward or backward. You are, in effect, freed from the directionality of time. You can set a breakpoint at a function or source line, even if it has already executed, and go back to it.

Example of Why Reverse Debugging Is Useful

Pretend that you have a memory corruption bug. Somehow, after your program has been running for a while, a pointer gets corrupted and the next time the pointer is used, your program crashes. But the pointer might be changed many times before the actual corruption occurs, and the code that corrupts it is not "supposed to" touch the pointer at all. Such bugs are notoriously hard to track down.

With reverse debugging you can now find that bug in one pass:

1 Record the program execution (including the crash).

2 Start the recording under `gdb` and `vmware-gdb-proxy`.

3 Run it until the crash.

4 Set a watchpoint on the corrupted pointer, and

5 Reverse-continue.

When the watchpoint triggers, that is where the pointer was corrupted. Problem solved (or at least, located).

Note: `gdb` version 7.0 (and later) also supports the `reverse-step` command, but `vmware-gdb-proxy` and Workstation do not support the `reverse-step` command at this time.

Replay Debugging with Shared Libraries

A common problem occurs when running `gdb` on one computer while debugging a program on a different computer, even a virtual computer. When the program you are debugging loads a shared library, `gdb` looks for the symbols of that library on its own local file system, which generally means that it finds the shared libraries (and symbols) for the host operating system, rather than those of the virtual guest operating system. Unless the host and guest OS are identical, this can lead to confusion.

The way to work around this problem is to make a copy of some (or all) of your virtual machine guest operating system's shared system libraries available in the host computer's file system, and then tell `gdb` where to find them.

This example shows how to manage some of the more common Linux system libraries. You might find that you have other libraries that you need to handle in the same way, depending on what packages are installed in your virtual machine.

First make a directory on the host file system:

```
host$ mkdir /home/username/guestos
host$ mkdir /home/username/guestos/usr
```

Then, by whatever means are convenient, copy the libraries from the guest to the host, preserving the directory hierarchy. This example shows use of `scp` from the guest:

```
guest$ scp -r /lib user@host:/home/usrname/guestos
guest$ scp -r /usr/lib user@host:/home/usrname/guestos/usr
```

Now, after starting `gdb` but before giving it the `target remote` command, type this `gdb` instruction:

```
(gdb) set solib-absolute-prefix /home/username/guestos
```

Now `gdb` looks in `/home/username/guestos/lib` instead of `/lib`.

Kernel Mode Debugging

Replay debugging is a convenient way to debug the Linux kernel.

With traditional kernel debugging, you must modify the kernel by including a `gdb` module such as `kgdb`, and setting up special interrupt request (IRQ) handlers. Even then, you cannot connect `gdb` to the kernel until serial drivers and IRQ handlers are initialized.

With replay debugging, you modify your kernel build simply by turning on debugging symbols in the build, and if you start making your recording before booting the guest machine, you can start debugging at the first line of debuggable code, typically `init/main.c:start_kernel()`.

You must have your kernel source and build tree mirrored on your host machine's file system (either by copying or sharing a directory) so that `gdb` can find the `vmlinux` kernel and source files.

Run `vmware-gdb-proxy` with only the `-V` (virtual machine) and `-K` (kernel mode) options. With these options, `vmware-gdb-proxy` pauses the recording at the very beginning, and waits for `gdb` to connect. For details, see "Command Line Options for vmware-gdb-proxy" on page 6.

Then run `gdb` in your source-build tree, giving it `vmlinux` for a symbol file. Use the usual `target remote` command to connect:

```
(gdb) target remote localhost:7732
```

You can now set a breakpoint at the function `start_kernel`, or at any other code that you would like to debug. You can debug I/O drivers, interrupt handlers, even the scheduler, provided the code is in the main kernel binary. (See below for debugging kernel modules). You can replay the recording as often as you like (with 100 percent determinism). If you are using `gdb` version 7.0 or later, you can even use reverse-continue to make the kernel run backward.

Debugging Linux Kernel Modules

Debugging a kernel module is a little more complicated, but not difficult.

As usual, you must make sure that both the binary executable file and the source files are available to `gdb` are on the host file system (either by copying or by sharing a directory).

Make a recording while your kernel module is installed and operating. You can begin the recording before the kernel itself boots, if required.

Step 1: After making the recording (and before rebooting the guest or uninstalling your kernel module), you must perform one additional step. Open a terminal in the guest, and type the following commands at the shell prompt. Make note of the addresses given for each section (`.text`, `.data`, `.bss`, and `.init.text`).

```
guest$ cd /sys/modules/<module_name>/sections
guest$ cat .text .data .bss .init.text
```

For example, assume that they are as follows:

```
* .text - 0x1111
* .data - 0x2222
* .bss  - 0x3333
* .init.text - 0x4444
```

Step 2: Choose a function in your kernel module where you would like to begin debugging, and use the function name to give the following shell command (also in the guest):

```
guest$ cat /proc/kallsyms | grep YourFunctionName
```

That address is used for your start address below. For example, assume that it is `0x12345678`.

Now suspend or power-off your virtual machine, go to a shell in your host machine, and run the `vmware-gdb-proxy` command as follows:

```
host$ vmware-gdb-proxy -V path-to-your-vm -S 0x12345678  -R your-recording-name -K
```

The guest recording starts, runs as far as the start-address you supplied (Step 2 above, the kernel module function that you want to start debugging), then pauses and waits for `gdb` to connect.

There is now a special command you must use to load the symbols for your kernel module into `gdb`.

Step3: First you start `gdb`, but without giving it any symbol file.

Then, assuming your kernel module symbol file is at `mydir/mymodule.ko`, and assuming the example values we obtained in Step 1 above, give `gdb` the following command:

```
(gdb) add-symbol-file mydir/mymodule.ko 0x1111 -s .data 0x2222 -s .bss 0x3333 -s .init.text
               0x4444
```

Now connect `gdb` to the virtual machine with the usual command, and you are ready to begin debugging your kernel module.

```
(gdb) target remote localhost:7732
```

Ending Your Replay Debugging Session

When finish a Replay Debugging session, there are two ways to end your `gdb` session: either `detach` or `kill`.

```
(gdb) detach
```

If you give `gdb` the detach command, `gdb` releases its connection to VMware Workstation, and the recording resumes playback. When the end of the recording is reached, the virtual machine "goes live" and you can either shut it down or continue interacting with it.

```
(gdb) kill
```

If you give `gdb` the kill command, `gdb` sends a message to the virtual machine causing it to power itself down. Then `gdb` drops the connection, and becomes ready to quit.

Learning More About Replay Debugging

If you want to learn more about the new Replay Debugging technology, compare notes with other users, and ask questions to the VMware staff and engineers, join the Replay Debugging Community Forum:

http://communities.vmware.com/community/vmtn/general/guestdebugmonitor

For replay debugging on Windows, see the *Integrated Virtual Debugger for Visual Studio Developer's Guide*.

Command Line Options for vmware-gdb-proxy

The following options are available for `vmvware-gdb-proxy`, as summarized in the usage message.

```
Usage: vmware-gdb-proxy <options>   Options:
    -h              help
    -v <verbosity>  verbosity level
    -V <vmx>        path to vmx file
```

```
-R <recording>  name of recording
-F <filename>   executable file to debug
-P <pid>        pid to debug (cannot be 999999.)
-S <addr>       address to start debugging (optional if -F is specified)
-K              debug kernel (not user process)
-A              attach to target process (if process was running when recording started)
-p              port number
-L              listen for gdb on local port only
```

More about these options:

- `-h` (help): Prints the above list of options.
- `-v` (verbosity): To be followed by a number from 0 to 2. Affects the amount of debugging messages that are logged in the log file. You might be asked to use this option if you file a bug report.
- `-V` (vmx): To be followed by the path to the VMX file for the virtual machine that you wish to debug.
- `-R` (recording name): To be followed by the name of the recording that you wish to debug.
- `-F` (executable file): To be followed by the path to the binary executable file that you wish to debug. The start address is derived by `vmware-gdb-proxy` when examining the file.
- `-P` (process id): To be followed by the PID (process ID) of the process in the virtual machine guest operating system that you wish to debug.
- `-n` (process name): To be followed by the name of the process in the virtual machine guest operating system that you wish to debug.
- `-S` (start address): To be followed by the address in the guest process at which you wish to begin debugging. The recording automatically plays up to this address before waiting for `gdb` to attach.
- `-K` (kernel mode): Use this option to debug in full system or kernel mode, rather than in process or user mode. All events in all processes on the guest OS (including the kernel) are made available to `gdb`.
- `-A` (attach to already running process): Use this option if the process that you wish to debug was already running when the recording started.
- `-p` (port / socket ID): To be followed by a port or socket ID, in case you do not wish to use the default socket ID (7732) for communication between `gdb` and `vmware-gdb-proxy`.
- `-L` (local port only): Use this option to tell `vmware-gdb-proxy` that it should listen for `gdb` to connect from a port or socket ID on the local host computer. By default, `vmware-gdb-proxy` allows `gdb` to connect from another host as well as from the local host.

Examples

To specify both the program to be debugged and its start address (`-F`), assuming that the executable is in `builds/Ubuntu/myprog` (in the host file system):

```
host$ vmware-gdb-proxy -F builds/Ubuntu/myprog [...]
```

To specify the program to be debugged by its process id in the guest operating system (`-P`) and give its starting address (`-S`):

```
host$ vmware-gdb-proxy -P 1234 -S 0x12345678 [...]
```

To specify the program to be debugged by its process name in the guest operating system (`-n`) and give its starting address (`-S`):

```
host$ vmware-gdb-proxy -n myprog -S 0x12345678 [...]
```

If you specify the program to be debugged by giving its guest process id or its guest process name, you must also specify the address where the guest process begins executing. This can be obtained by loading the binary executable into `gdb` and typing:

```
(gdb) print _start
```

It is simpler to use the `-F` option to specify the binary executable in the host file system: in that case, you do not need to specify the start address.

Known Limitations

Replay Debugging on Linux is subject to the following limitations:

- Replay Debugging is not supported with 64-bit versions of VMware Workstation, or with 64-bit guest operating systems.
- Reverse Debugging does not support `reverse-step`, `reverse-stepi`, or any `gdb` commands that depend on being able to step in reverse.
- While debugging a recording, you cannot modify program registers, variables, or memory in the recorded guest program. You also cannot modify the sequence of execution in the guest (for example with `gdb`'s `jump` or `return` commands).
- When starting a session with `vmware-gdb-proxy`, you cannot specify the guest process id "999999" or the guest process name "invalid."

Linux Kernel Offsets

Find the Linux kernel version in the first column of Table 1, and copy the `debugStub.linuxOffsets` in the second column to customize the VMX file for that guest virtual machine.

Table 1. Linux Kernel Offsets

Linux Kernel Version	debugStub.linuxOffsets
2.4.20-31.9bigmem	0x20414,0x6c,0,0x54,0x3fa,0x8c,0x430,0x10,0x18,0x24,0x2000,0,0,0x10
2.4.20-31.9	0x20414,0x6c,0,0x54,0x2fe,0x8c,0x330,0x10,0x18,0x24,0x2000,0,0,0x10
2.4.20-31.9smp	0x20414,0x6c,0,0x54,0x3fa,0x8c,0x430,0x10,0x18,0x24,0x2000,0,0,0x10
2.4.21-20.ELhugemem	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-20.EL	0x20415,0x78,0,0x60,0x358,0x98,0x380,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-20.ELsmp	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-278-athlon	0x20415,0x58,0x50,0,0x24a,0x80,0x280,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-278-default	0x20415,0x58,0x50,0,0x24a,0x80,0x280,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-278-psmp	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-278-smp	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-27.ELhugemem	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-27.EL	0x20415,0x78,0,0x60,0x358,0x98,0x380,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-27.ELsmp	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-306-athlon	0x20415,0x58,0x50,0,0x24a,0x80,0x280,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-306-debug	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-306-default	0x20415,0x58,0x50,0,0x24a,0x80,0x280,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-306-psmp	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-306-smp4G	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-306-smp	0x20415,0x58,0x50,0,0x346,0x80,0x370,0x10,0,0xc,0x2000,0,0,0x10
2.4.21-32.ELsmp	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-32.EL	0x20415,0x78,0,0x60,0x358,0x98,0x380,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-32.ELhugemem	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-32.ELsmp	0x20415,0x78,0,0x60,0x54c,0x98,0x580,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.21-32.EL	0x20415,0x78,0,0x60,0x358,0x98,0x380,0x14,0x20,0x2c,0x2000,0,0,0x10
2.4.9-e.40enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.40	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.4.9-e.40smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.48enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.48	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.48smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.49enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.49	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.49smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.49summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.57enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.57	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.57smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.62	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.65BOOT	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.65	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.65enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.65smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.65	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68BOOT	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.68	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70BOOT	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.70	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71BOOT	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.71	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.72enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.4.9-e.72smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.72summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.72	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.74enterprise	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.74smp	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.74summit	0x20409,0x58,0x4c,0,0x336,0x74,0x360,0xc,0,0xc,0x2000,0,0,0x10
2.4.9-e.74	0x20409,0x58,0x4c,0,0x23a,0x74,0x260,0xc,0,0xc,0x2000,0,0,0x10
2.6.10-5-386	0x2060a,0x70,0,0x58,0x1ae,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.10-v	0x2060a,0x70,0,0x58,0x1ae,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.10-5-686-smp	0x2060a,0x70,0,0x58,0x1b6,0x94,0x1e0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.10-5-686	0x2060a,0x70,0,0x58,0x1ae,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-20a-bigsmp	0x2060b,0x7c,0,0x64,0x1b8,0xa0,0x1e0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-20a-default	0x2060b,0x70,0,0x58,0x1a8,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-20a-smp	0x2060b,0x70,0,0x58,0x1ac,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.7-bigsmp	0x2060b,0x7c,0,0x64,0x1b8,0xa0,0x1e0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.7-default	0x2060b,0x70,0,0x58,0x1a8,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.7-smp	0x2060b,0x70,0,0x58,0x1ac,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.9-bigsmp	0x2060b,0x7c,0,0x64,0x1b8,0xa0,0x1e0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.9-default	0x2060b,0x70,0,0x58,0x1a8,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.11.4-21.9-smp	0x2060b,0x70,0,0x58,0x1ac,0x94,0x1d0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-12mdkenterprise	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-12mdksmp	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-12mdk-i586-up-1GB	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-12mdk-i686-up-4GB	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-26mdk	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-26mdksmp	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-26mdk-i586-up-1GB	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-26mdk-i686-up-4GB	0x2060c,0x74,0,0x5c,0x1a4,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-9-k7-smp	0x2060c,0x74,0,0x5c,0x1a0,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-9-k7	0x2060c,0x74,0,0x5c,0x1a0,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-9-386	0x2060c,0x74,0,0x5c,0x1a0,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-9-686-smp	0x2060c,0x74,0,0x5c,0x1a0,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.12-9-686	0x2060c,0x74,0,0x5c,0x1a0,0x98,0x1c0,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.13-15-bigsmp	0x2060d,0x84,0,0x6c,0x1b0,0xa8,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.13-15-default	0x2060d,0x78,0,0x60,0x1a4,0x9c,0x1c0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.13-15-smp	0x2060d,0x78,0,0x60,0x1a4,0x9c,0x1c0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-23-386	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-23-k7	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-23-server	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-23-686	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-27-k7	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.6.15-27-686	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.15-27-386	0x2060f,0x78,0,0x60,0x1ac,0x9c,0x1d0,0x20,0x18,0x28,0x2000,0,0,0x10
2.6.16.13-4-bigsmp	0x20610,0x98,0,0x80,0x1c4,0xbc,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.13-4-default	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.13-4-smp	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-bigsmp	0x20610,0x98,0,0x80,0x1c4,0xbc,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-debug	0x20610,0x98,0,0x80,0x1c4,0xbc,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-default	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-kdump	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-smp	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-xenpae	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.21-0.8-xen	0x20610,0x8c,0,0x74,0x1b8,0xb0,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.46-0.12-bigsmp	0x20610,0x9c,0,0x84,0x1c8,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.46-0.12-default	0x20610,0x90,0,0x78,0x1bc,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.46-0.12-smp	0x20610,0x90,0,0x78,0x1bc,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.21-bigsmp	0x20610,0x9c,0,0x84,0x1d0,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.21-default	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.21-smp	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.21-vmipae	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.21-vmi	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.23-bigsmp	0x20610,0x9c,0,0x84,0x1d0,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.23-default	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.23-smp	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.23-vmipae	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.16.60-0.23-vmi	0x20610,0x90,0,0x78,0x1c4,0xb4,0x1e0,0x24,0x18,0x28,0x2000,0,0,0x10
2.6.17-10-386	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-10-server-bigiron	0x20611,0x80,0,0x68,0x194,0xa4,0x1b0,0x24,0x18,0x28,0x2000,0xc4,0xec,0x10
2.6.17-10-generic	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-10-server	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdv	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdventerprise	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdvlegacy	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.12-12mdksmp	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdv	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdventerprise	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.17-5mdvlegacy	0x20611,0x7c,0,0x64,0x190,0xa0,0x1b0,0x24,0x18,0x28,0x2000,0xc0,0xe8,0x10
2.6.18.2-34-bigsmp	0x20612,0xa4,0,0x8c,0x1b8,0xc8,0x1e0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.18.2-34-default	0x20612,0x98,0,0x80,0x1ac,0xbc,0x1d0,0x24,0x18,0x28,0x2000,0xdc,0x104,0x10
2.6.18-53.11AXS3PAE	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-53.11AXS3	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-6-686-bigmem	0x20612,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.6.18-6-k7	0x20612,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.18-6-686	0x20612,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.18-8.10AXPAE	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-8.10AX	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-8.10AXxen	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-8.el5PAE	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-8.el5	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.18-8.el5xen	0x20612,0x84,0,0x7c,0x194,0xa8,0x1b0,0x28,0x18,0x28,0x1000,0xc4,0xec,0x10
2.6.20-15-server-bigiron	0x20614,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.20-15-generic	0x20614,0x80,0,0x68,0x194,0xa4,0x1b0,0x24,0x18,0x28,0x2000,0xc4,0xec,0x10
2.6.20-15-server	0x20614,0x80,0,0x68,0x194,0xa4,0x1b0,0x24,0x18,0x28,0x2000,0xc4,0xec,0x10
2.6.22-14-generic	0x20616,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.22-14-server	0x20616,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.22-14-ume	0x20616,0x84,0,0x6c,0x198,0xa8,0x1c0,0x24,0x18,0x28,0x2000,0xc8,0xf0,0x10
2.6.22.5-31-bigsmp	0x20616,0xa4,0,0x8c,0x1b8,0xc8,0x1e0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.22.5-31-default	0x20616,0x98,0,0x80,0x1ac,0xbc,0x1d0,0x24,0x18,0x28,0x2000,0xdc,0x104,0x10
2.6.22.9-desktop-1mdv	0x20616,0xb4,0,0x9c,0x1c8,0xd8,0x1f0,0x24,0x18,0x28,0x2000,0xf8,0x120,0x10
2.6.22.9-laptop-1mdv	0x20616,0xb4,0,0x9c,0x1c8,0xd8,0x1f0,0x24,0x18,0x28,0x2000,0xf8,0x120,0x10
2.6.22.9-server-1mdv	0x20616,0xb4,0,0x9c,0x1c8,0xd8,0x1f0,0x24,0x18,0x28,0x2000,0xf8,0x120,0x10
2.6.24-16-generic	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-16-server	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-16-virtual	0x20618,0x9c,0,0x84,0x1c5,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0xe0,0x108,0x10
2.6.24-19-generic	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-19-server	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-19-virtual	0x20618,0x9c,0,0x84,0x1c5,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0xe0,0x108,0x10
2.6.24-23-generic	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-23-server	0x20618,0xa4,0,0x8c,0x1cd,0xc8,0x1f0,0x24,0x18,0x28,0x2000,0xe8,0x110,0x10
2.6.24-23-virtual	0x20618,0x9c,0,0x84,0x1c5,0xc0,0x1f0,0x24,0x18,0x28,0x2000,0xe0,0x108,0x10
2.6.25.5-1.1-default	0x20619,0x1b0,0,0x198,0x2ed,0x1d4,0x310,0x24,0x18,0x28,0x2000,0x1f4,0x21c,0x10
2.6.25.5-1.1-pae	0x20619,0x1bc,0,0x1a4,0x2f9,0x1e0,0x320,0x24,0x18,0x28,0x2000,0x200,0x228,0x10
2.6.26-1-686-bigmem	0x2061a,0xec,0,0xd4,0x229,0x110,0x250,0x24,0x18,0x28,0x2000,0x130,0x158,0x10
2.6.26-1-686	0x2061a,0xec,0,0xd4,0x229,0x110,0x250,0x24,0x18,0x28,0x2000,0x130,0x158,0x10
2.6.27.5-117.fc10.i686	0x2061b,0x1c8,0,0x1c0,0x314,0x1ec,0x338,0x28,0x18,0x28,0x1000,0x20c,0x244,0x10
2.6.27-desktop586-0.rc8.2mnb	0x2061b,0x1b0,0,0x1a8,0x2fc,0x1d4,0x318,0x24,0x18,0x28,0x2000,0x1f4,0x22c,0x10
2.6.27-desktop-0.rc8.2mnb	0x2061b,0x1b0,0,0x1a8,0x2fc,0x1d4,0x318,0x24,0x18,0x28,0x2000,0x1f4,0x22c,0x10
2.6.27-server-0.rc8.2mnb	0x2061b,0x1b0,0,0x1a8,0x2fc,0x1d4,0x318,0x24,0x18,0x28,0x2000,0x1f4,0x22c,0x10
2.6.27.19-3.1-default	0x2061b,0x1c8,0,0x1c0,0x314,0x1ec,0x330,0x24,0x18,0x28,0x2000,0x20c,0x244,0x10
2.6.27.19-3.1-pae	0x2061b,0x1d4,0,0x1cc,0x320,0x1f8,0x33c,0x24,0x18,0x28,0x2000,0x218,0x250,0x10
2.6.27.19-3.1-vmi	0x2061b,0x1c8,0,0x1c0,0x314,0x1ec,0x330,0x24,0x18,0x28,0x2000,0x20c,0x244,0x10
2.6.27.7-9-default	0x2061b,0x1c8,0,0x1c0,0x314,0x1ec,0x330,0x24,0x18,0x28,0x2000,0x20c,0x244,0x10
2.6.27.7-9-pae	0x2061b,0x1d4,0,0x1cc,0x320,0x1f8,0x33c,0x24,0x18,0x28,0x2000,0x218,0x250,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.6.27-7-generic	0x2061b,0x1cc,0,0x1c4,0x318,0x1f0,0x33c,0x24,0x18,0x28,0x2000,0x210,0x248,0x10
2.6.27-7-server	0x2061b,0x1cc,0,0x1c4,0x318,0x1f0,0x33c,0x24,0x18,0x28,0x2000,0x210,0x248,0x10
2.6.3-7mdkenterprise	0x20603,0x68,0,0x50,0x30a,0x88,0x330,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.3-7mdk	0x20603,0x68,0,0x50,0x302,0x88,0x330,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.3-7mdksmp	0x20603,0x68,0,0x50,0x30a,0x88,0x330,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.3-7mdk-i686-up-4GB	0x20603,0x68,0,0x50,0x302,0x88,0x330,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.139-bigsmp	0x20605,0x7c,0,0x64,0x28d,0x9c,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.139-default	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.139-smp	0x20605,0x70,0,0x58,0x281,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.145-bigsmp	0x20605,0x7c,0,0x64,0x28d,0x9c,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.145-debug	0x20605,0x7c,0,0x64,0x295,0x9c,0x2c0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.145-default	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.145-SLRS	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.145-smp	0x20605,0x70,0,0x58,0x281,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.155.29-bigsmp	0x20605,0x7c,0,0x64,0x28d,0x9c,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.155.29-default	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.155.29-smp	0x20605,0x70,0,0x58,0x281,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.191-bigsmp	0x20605,0x7c,0,0x64,0x28d,0x9c,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.191-default	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.191-smp	0x20605,0x70,0,0x58,0x281,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.201-bigsmp	0x20605,0x7c,0,0x64,0x28d,0x9c,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.201-default	0x20605,0x70,0,0x58,0x279,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.201-smp	0x20605,0x70,0,0x58,0x281,0x90,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.244-bigsmp	0x20605,0x7c,0,0x64,0x291,0xa0,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.244-debug	0x20605,0x7c,0,0x64,0x299,0xa0,0x2c0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.244-default	0x20605,0x70,0,0x58,0x27d,0x94,0x2a0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.244-smp	0x20605,0x70,0,0x58,0x285,0x94,0x2b0,0x10,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.308-bigsmp	0x20605,0x7c,0,0x64,0x279,0xa0,0x2a0,0x14,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.308-default	0x20605,0x70,0,0x58,0x265,0x94,0x290,0x14,0x18,0x28,0x2000,0,0,0x10
2.6.5-7.308-smp	0x20605,0x70,0,0x58,0x26d,0x94,0x290,0x14,0x18,0x28,0x2000,0,0,0x10
2.6.8-1smp64G	0x20608,0x74,0,0x5c,0x26e,0x94,0x290,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-1smp	0x20608,0x68,0,0x50,0x262,0x88,0x280,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-1	0x20608,0x68,0,0x50,0x25a,0x88,0x280,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.14-bigsmp	0x20608,0x74,0,0x5c,0x220,0x94,0x240,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.14-default	0x20608,0x68,0,0x50,0x20c,0x88,0x230,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.14-smp	0x20608,0x68,0,0x50,0x214,0x88,0x230,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.18-bigsmp	0x20608,0x74,0,0x5c,0x220,0x94,0x240,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.18-debug	0x20608,0x88,0,0x70,0x23c,0xa8,0x260,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.18-default	0x20608,0x68,0,0x50,0x20c,0x88,0x230,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.8-24.18-smp	0x20608,0x68,0,0x50,0x214,0x88,0x230,0x1c,0x18,0x28,0x2000,0,0,0x10
2.6.9-11.ELhugemem	0x20609,0x70,0,0x58,0x226,0x94,0x250,0x20,0x1c,0x2c,0x1000,0,0,0x10

Table 1. Linux Kernel Offsets (Continued)

Linux Kernel Version	debugStub.linuxOffsets
2.6.9-11.ELsmp	0x20609,0x70,0,0x58,0x226,0x94,0x250,0x20,0x1c,0x2c,0x1000,0,0,0x10
2.6.9-11.EL	0x20609,0x70,0,0x58,0x246,0x94,0x270,0x20,0x1c,0x2c,0x1000,0,0,0x10
2.6.9-5.ELhugemem	0x20609,0x70,0,0x58,0x226,0x94,0x250,0x20,0x1c,0x2c,0x1000,0,0,0x10
2.6.9-5.EL	0x20609,0x70,0,0x58,0x246,0x94,0x270,0x20,0x1c,0x2c,0x1000,0,0,0x10
2.6.9-5.ELsmp	0x20609,0x70,0,0x58,0x226,0x94,0x250,0x20,0x1c,0x2c,0x1000,0,0,0x10

Determining Kernel Offsets

If you do not find your kernel version in Appendix A, or if your kernel is customized, you need to get the kernel offsets by running a small program. First, get the kernel source code and build it by using `make` in the kernel source directory. Second, cut and paste the content below into three files on your Linux virtual machine, then run script `getlinuxoffsets_x86` to get the offsets.

Command-Line Syntax

```
getlinuxoffsets_x86 [<path_to_include_file> [<version_number>]]
```

- `<path_to_include_file>` is the path to the kernel header file directory. If not specified, defaults to `/usr/src/linux`.
- `<version_number>` is the version number of your kernel. This option is needed only if you are using a customized 2.4.21-278 or 2.4.21-306 kernel.

File getlinuxoffsets1.c

```
/* Copyright 2007 VMware, Inc.  All rights reserved. */
#include <stdio.h>
extern unsigned offsets[];
extern unsigned offsets_cnt;
int
main(int argc, char **argv)
{
   unsigned i;
   printf("# Linux kernel version: %s\n", argv[1]);

#ifdef DEBUG
   printf("# Include path: %s\n", argv[2]);
#endif

   printf("debugStub.linuxOffsets = \"");
   for (i = 0; i < offsets_cnt; i++) {
      printf("%#x%c", offsets[i], (i == offsets_cnt - 1) ? '\"' : ',');
   }
   printf("\n");
   return 0;
}
```

File getlinuxoffset2.c

```
/* Copyright 2007 VMware, Inc.  All rights reserved. */
#define __KERNEL__              1
#define MODULE                  1

#include <linux/version.h>
#include <linux/autoconf.h>
#include <linux/types.h>

#ifndef KBUILD_BASENAME
#define KBUILD_BASENAME "debugstub"
```

```
#endif
#include <linux/sched.h>

#define OFFS(_st, _fld) ((unsigned)&((struct _st *)0)->_fld)
#define NELEM(_arr) (sizeof(_arr) / sizeof(_arr[0]))

unsigned offsets[] = {
   LINUX_VERSION_CODE,
   OFFS(task_struct, mm),
#if FORCE_NEXT_TASK || LINUX_VERSION_CODE < KERNEL_VERSION(2,4,20)
   OFFS(task_struct, next_task),
   0,
#else
   0,
   OFFS(task_struct, tasks),
#endif
   OFFS(task_struct, comm),
   OFFS(task_struct, pid),
   OFFS(task_struct, thread),
   OFFS(mm_struct, pgd),
#if CONFIG_X86_64
   OFFS(thread_struct, rsp0),
#else
#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,25)
   OFFS(thread_struct, esp0),
#else
   OFFS(thread_struct, sp0),
#endif
#endif
   OFFS(thread_struct, fs),
   THREAD_SIZE,
#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,17)
   0,
   0,
#else
   OFFS(task_struct, group_leader),
   OFFS(task_struct, thread_group),
#endif
   sizeof ((struct task_struct *)0)->comm,
};
unsigned offsets_cnt = NELEM(offsets);
```

Script getlinuxoffsets_x86

```
#!/bin/bash
# Copyright 2007 VMware, Inc.  All rights reserved.

# Set the compiler to be used for getting the offsets.
if [ "$CC" == "" ]; then
   CC=gcc
   echo "# Using gcc as the compiler. If you want to use another compiler, set the environment
               variable CC"
fi

# Set the path from where to get the headers. If the user does not specify one,
# use /usr/src/linux/include
if [ "$1" == "" ]; then
   INCLUDE_PATH=/usr/src/linux/include
   echo "# Using linux kernel headers from /usr/src/linux/include"
else
   INCLUDE_PATH="$1/"
   echo "# Using linux kernel headers from" $1
fi

# We use the version of the kernel for deciding the offset of the next task
# from the task_struct. See below for more details.
# If there is a properties file, use it to get the kernel version. Otherwise
# use the one provided on the command line.
```

```
if [ "$2" == "" ]; then
   VERSION=`grep UtsRelease $INCLUDE_PATH/properties | cut -d' ' -f2`
else
   VERSION="$2"
fi

# The linux kernel version 2.4.21 has some conflicts with respect to the
# 'next_task' and 'tasks' fields in the struct task_struct.
# Two versions; viz. 2.4.21-278 and 2.4.21-306 have the 'next_task' field in the
# strcut task_struct, wheres others name the same field as 'tasks'.

# if the version is 2.4.21-278 or 2.4.21-306, set the FORCE_NEXT_TASKS flag
VERSION_2_4_21_278="2.4.21-278"
OFFSET=`echo "$VERSION" | awk -v a="$VERSION_2_4_21_278" '{ print index($0,a)}'`

VERSION_2_4_21_306="2.4.21-306"
OFFSET2=`echo "$VERSION" | awk -v a="$VERSION_2_4_21_306" '{ print index($0,a)}'`

if [ "$OFFSET" -eq "1" -o "$OFFSET2" -eq "1" ]
then
   FORCE_FLAGS="FORCE_NEXT_TASK"
else
   FORCE_FLAGS="NONE"
fi

$CC -c -D "$FORCE_FLAGS" -I "$INCLUDE_PATH" -I "$INCLUDE_PATH"/asm/mach-default \
   getlinuxoffsets2.c && \
$CC -o getlinuxoffsets.tmp -D "$FORCE_FLAGS" getlinuxoffsets1.c getlinuxoffsets2.o && \
./getlinuxoffsets.tmp $VERSION $INCLUDE_PATH && \
rm -f getlinuxoffsets.tmp getlinuxoffsets1.o getlinuxoffsets2.o
```