DebuggingKernelOops

Debugging Central This page is part of the debugging series — pages with debugging details for a variety of Ubuntu packages.

There are times when the code in the kernel will have errors and result in a system fault. This is known as a kernel Oops. The kernel Oops report is typically dumped to the console as well as output to a log file like /var/log/dmesg . Although a system fault may not render a machine useless, it will require a reboot as the machine may be left in an unusable/inconsistent state. The information contained here can be helpful with debugging a kernel Oops report. It will also help educate a bug reporter in gathering appropriate information to supply to the kernel team to help resolve system faults.

Contents

Kernel Oops Report

An example of a kernel Oops report looks like the following:

```
BUG: unable to handle kernel NULL pointer dereference at virtual address 00000008
 printing eip:
c022a7b5
*pde = 00000000
Oops: 0000 [#1]
SMP
Modules linked in: thinkpad_acpi ppdev speedstep_lib cpufreq_conservative cpufreq_userspace cpufreq_ondemand cpufreq_stats cpu
CPU:    0
EIP:    0060:[<c022a7b5>]    Not tainted VLI
EFLAGS: 00010202   (2.6.22-12-generic #1)
EIP is at acpi_ns_internalize_name+0xd/0x83
eax: 00000008   ebx: 00000000   ecx: 00000000   edx: c7879e54
esi: d0b980c0   edi: c7879e54   ebp: c7879e70   esp: c7879de8
ds: 007b   es: 007b   fs: 00d8  gs: 0033  ss: 0068
Process modprobe (pid: 4467, ti=c7878000 task=ce5c94c0 task.ti=c7878000)
Stack: 00000000 00000000 d0b97e60 00008080 c01c4390 d0b97e60 00000000 00000000
       d0b980c0 00000000 c7879e70 c022a85c d0b97e60 c795d030 c7c604e0 c01c44ef
       00000004 d0b97e60 c7acea18 c01c3884 00008080 00000004 00000004 00000080
Call Trace:
 [<c01c4390>] __sysfs_new_dirent+0x20/0x50
 [<c022a85c>] acpi_ns_get_node+0x31/0x93
 [<c01c44ef>] sysfs_make_dirent+0x2f/0x50
 [<c01c3884>] sysfs_add_file+0x74/0x90
 [<d0b910b7>] drv_acpi_handle_init+0x37/0x90 [thinkpad_acpi]
 [<c0231aef>] acpi_ut_release_mutex+0x5b/0x63
 [<c0233ac0>] acpi_method_notify_enable+0x15/0x34
 [<d0b5ba32>] cmos_init+0x52/0x70 [thinkpad_acpi]
 [<d0b5c21f>] thinkpad_acpi_module_init+0x27f/0x69a [thinkpad_acpi]
 [<c014a811>] sys_init_module+0x151/0x1a00
 [<c01fb8cf>] prio_tree_insert+0x1f/0x250
 [<c01041d2>] sysenter_past_esp+0x6b/0xa9
 =======================
Code: c7 44 24 14 01 00 00 00 8b 54 24 14 8d 04 96 e9 f2 fe ff ff 83 c4 18 89 d0 5b 5e 5f 5d c3 55 57 89 d7 56 53 83 ec 1c 85
EIP: [<c022a7b5>] acpi_ns_internalize_name+0xd/0x83 SS:ESP 0068:c7879de8
```

Debugging a Kernel Oops

```
BUG: unable to handle kernel NULL pointer dereference at virtual address 00000008
```

The above type of Oops is one that is commonly encountered. This means that we've dereferenced an invalid pointer which resulted in the kernel Oops. The rest of the Oops report displays the status of the system at the time of the fault.

```
Oops: 0000 [#1]
```

Many times one Oops will trigger multiple Oopses. When debugging multiple Oops reports, be sure to start with the very first one.

```
EIP is at acpi_ns_internalize_name+0xd/0x83
```

Usually, the most relevant information we can gather from the Oops in this situation will be the EIP and address of the bad call. For 64 bit users, you'll want to look at the RIP instead. The EIP/RIP usually indicates where the problem occurred. In this case we can see that at 0xd bytes into the function acpi_ns_internalize_name we hit the Oops. This will give a good indication of where to start looking for the bad code.

```
EIP:    0060:[<c022a7b5>]    Not tainted VLI
```

Tainted Kernels

This is also a good time to mention Tainted Kernels. Some Oops reports contain the string 'Tainted: ' after the EIP. This indicates that the kernel has been tainted by some mechanism. 'Tainted: ' is followed by a series of position-sensitive characters, each representing a particular tainted value. If the letter does not appear the kernel is not tainted in that fashion.

1. 'G' if all modules loaded have a GPL or compatible license

2. 'P' if any proprietary module has been loaded. Modules without a MODULE_LICENSE or with a MODULE_LICENSE that is not recognised by insmod as GPL compatible are assumed to be proprietary.
3. 'F' if any module was force loaded by "insmod -f".
4. 'S' if the Oops occurred on an SMP kernel running on hardware that hasn't been certified as safe to run multiprocessor. Currently this occurs only on various Athlons that are not SMP capable.
5. 'R' if a module was force unloaded by "rmmod -f".
6. 'M' if any processor has reported a Machine Check Exception.
7. 'B' if a page-release function has found a bad page reference or some unexpected page flags.
8. 'U' if a user or user application specifically requested that the Tainted flag be set.
9. 'D' if the kernel has died recently, i.e. there was an OOPS or BUG.
10. 'W' if a warning has previously been issued by the kernel.
11. 'C' if a staging module / driver has been loaded.
12. 'I' if the kernel is working around a sever bug in the platform's firmware (BIOS or similar).

An example output of a tainted line:

```
[187487.973317] Pid: 810, comm: iwl3945 Tainted: P   M        2.6.35-22-generic #33-Ubuntu
```

The primary reason for the 'Tainted: ' string is to tell kernel debuggers if this is a clean kernel or if anything unusual has occurred. Tainting is permanent: even if an offending module is unloaded, the tainted value remains to indicate that the kernel is not trustworthy.

In the event that you find a tainted value not listed here you can look it up in Documentation/oops-tracing.txt of the kernel source code which is in the kernel source package.

Call Stack Details

```
Stack: 00000000 00000000 d0b97e60 00008080 c01c4390 d0b97e60 00000000 00000000
        d0b980c0 00000000 c7879e70 c022a85c d0b97e60 c795d030 c7c604e0 c01c44ef
        00000004 d0b97e60 c7acea18 c01c3884 00008080 00000004 00000004 00000080
 Call Trace:
  [<c01c4390>] __sysfs_new_dirent+0x20/0x50
  [<c022a85c>] acpi_ns_get_node+0x31/0x93
  [<c01c44ef>] sysfs_make_dirent+0x2f/0x50
  [<c01c3884>] sysfs_add_file+0x74/0x90
  [<d0b910b7>] drv_acpi_handle_init+0x37/0x90 [thinkpad_acpi]
  [<c0231aef>] acpi_ut_release_mutex+0x5b/0x63
  [<c0233ac0>] acpi_method_notify_enable+0x15/0x34
  [<d0b5ba32>] cmos_init+0x52/0x70 [thinkpad_acpi]
  [<d0b5c21f>] thinkpad_acpi_module_init+0x27f/0x69a [thinkpad_acpi]
  [<c014a811>] sys_init_module+0x151/0x1a00
  [<c01fb8cf>] prio_tree_insert+0x1f/0x250
  [<c01041d2>] sysenter_past_esp+0x6b/0xa9
```

Sometimes you'll need to derive even more information from the Call Stack. This trace shows you how you got to where the code broke. You'll notice that **[thinkpad_acpi]** trails some of the Call Trace output. This indicates the module for which the code of the calling routine resides in. This is often helpful in narrowing down which module contains the issue.

Submitting a Kernel Oops Bug Report

Learning to debug and fix a kernel Oops is not an easy task for the average user. However, any user should be able to gather the appropriate information to help the kernel team debug and fix the issue. If you ever witness a kernel Oops please file a linux bug report in Launchpad. The following steps help guide you through this reporting process:

1. Search to see if someone else has already reported this issue. However, make sure you are experiencing the _same_ Oops (ie Call Trace looks the same, EIP is at the same location, you have similar hardware, you use the same steps to reproduce the Oops, etc.)
   - If you are convinced the report is the same, please add any further useful information to the existing bug report. It will also help if you subscribe youself to that report so you are notified of any updates regarding the bug.
   - If the report is not the same Oops you are experiencing, please proceed to Step 2.
2. Make sure the report is filed against the proper package. For a kernel Oops this will be 'linux'.
3. Please be sure to tag your report as 'kernel-oops'. This can be done be clicking on the "Edit description/tags" link and entering the tag 'kernel-oops' into the Tags field.
4. Please provide a descriptive Subject / Title for your bug.
   - BAD example - "Kernel Oops"
   - GOOD example - "Kernel Oops - unable to handle kernel NULL pointer dereference; EIP is at acpi_ns_internalize_name+0xd/0x83"
5. Please provide the kernel team as much information as possible:
   - If you are able to consistently trigger the kernel Oops, please document the exact steps to reproduce it.
   - Capture the full output of the kernel Oops as well as any output that leads up to the kernel Oops. This can usually be found in the dmesg output.
     - `dmesg > dmesg.log`
     - attach dmesg.log to the bug report
   - Also attach the following information:
     - `uname -a > uname-a.log`
     - `cat /proc/version_signature > version.log`
     - `sudo lspci -vvnn > lspci-vvnn.log`

CategoryBugSquad CategoryDebugging

DebuggingKernelOops (last edited 2011-03-02 18:33:33 by https://login.launchpad.net/+id/HAnJ3rt @ c-98-246-63-231.hsd1.wa.comcast.net[98.246.63.231]:brian-murray)