Chinese translated version of Documentation/oops-tracing.txt

If you have any comment or update to the content, please contact the original document maintainer directly. However, if you have a problem communicating in English you can also ask the Chinese maintainer for help. Contact the Chinese maintainer if this translation is outdated or if there is a problem with the translation.

Chinese maintainer: Dave Young <hidave.darkstar@gmail.com>

Documentation/oops-tracing.txt 的中文翻译

如果想评论或更新本文的内容，请直接联系原文档的维护者。如果你使用英文交流有困难的话，也可以向中文版维护者求助。如果本翻译更新不及时或者翻译存在问题，请联系中文版维护者。

中文版维护者： 杨瑞 Dave Young <hidave.darkstar@gmail.com>
中文版翻译者： 杨瑞 Dave Young <hidave.darkstar@gmail.com>
中文版校译者： 李阳 Li Yang <leo@zh-kernel.org>
王聪 Wang Cong <xiyou.wangcong@gmail.com>

以下为正文

注意： ksymoops 在2.6中是没有用的。 请以原有格式使用Oops(来自dmesg，等等)。忽略任何这样那样关于“解码Oops”或者“通过ksymoops运行”的文档。 如果你贴出运行过
ksymoops的来自2.6的Oops，人们只会让你重贴一次。

快速总结

发现Oops并发送给看似相关的内核领域的维护者。别太担心对不上号。如果你不确定就发给和你所做的事情相关的代码的负责人。 如果可重现试着描述怎样重构。 那甚至比oops更有价值。

如果你对于发送给谁一无所知， 发给linux-kernel@vger.kernel.org。感谢你帮助Linux尽可能地稳定。

Oops在哪里?

通常Oops文本由klogd从内核缓冲区里读取并传给syslogd，由syslogd写到syslog文件中，典型地是/var/log/messages(依赖于/etc/syslog.conf)。有时klogd崩溃了,这种情况下你能够运行dmesg > file来从内核缓冲区中读取数据并保存下来。 否则你可以
cat /proc/kmsg > file， 然而你必须介入中止传输， kmsg是一个“永不结束的文件”。如
果机器崩溃坏到你不能输入命令或者磁盘不可用那么你有三种选择:-

（1） 手抄屏幕上的文本待机器重启后再输入计算机。 麻烦但如果没有针对崩溃的准备，这是仅有的选择。 另外，你可以用数码相机把屏幕拍下来-不太好，但比没有强。 如果信息滚动到了终端的上面，你会发现以高分辨率启动（比如，vga=791）会让你读到更多的文本。（注意：这需要vesafb，所以对‘早期’的oops没有帮助）

（2）用串口终端启动（请参看Documentation/serial-console.txt），运行一个null modem到另一台机器并用你喜欢的通讯工具获取输出。Minicom工作地很好。

（3）使用Kdump（请参看Documentation/kdump/kdump.txt），
使用在Documentation/kdump/gdbmacros.txt中定义的dmesg gdb宏，从旧的内存中提取内核环形缓冲区。

完整信息

注意：以下来自于Linus的邮件适用于2.4内核。 我因为历史原因保留了它，并且因为其中一些信息仍然适用。 特别注意的是，请忽略任何ksymoops的引用。

From: Linus Torvalds <torvalds@osdl.org>

怎样跟踪Oops.. [原发到linux-kernel的一封邮件]

主要的窍门是有五年和这些烦人的oops消息打交道的经验;-)

实际上，你有办法使它更简单。我有两个不同的方法：

```
gdb /usr/src/linux/vmlinux
gdb> disassemble <offending_function>
```

那是发现问题的简单办法，至少如果bug报告做的好的情况下（象这个一样-运行ksymoops得到oops发生的函数及函数内的偏移）。

哦，如果报告发生的内核以相同的编译器和相似的配置编译它会有帮助的。

另一件要做的事是反汇编bug报告的“Code”部分：ksymoops也会用正确的工具来做这件事，
但如果没有那些工具你可以写一个傻程序：

```
char str[] = "\xXX\xXX\xXX...";
main(){}
```

并用gcc -g编译它然后执行“disassemble str”（XX部分是由Oops报告的值-你可以仅剪切粘贴并用“\x”替换空格-我就是这么做的，因为我懒得写程序自动做这一切）。

另外，你可以用scripts/decodecode这个shell脚本。它的使用方法是：
decodecode < oops.txt

“Code”之后的十六进制字节可能（在某些架构上）有一些当前指令之前的指令字节以及当前和之后的指令字节

```
Code: f9 0f 8d f9 00 00 00 8d 42 0c e8 dd 26 11 c7 a1 60 ea 2b f9 8b 50 08 a1
64 ea 2b f9 8d 34 82 8b 1e 85 db 74 6d 8b 15 60 ea 2b f9 <8b> 43 04 39 42 54
7e 04 40 89 42 54 8b 43 04 3b 05 00 f6 52 c0
```

最后，如果你想知道代码来自哪里，你可以：

```
cd /usr/src/linux
make fs/buffer.s        # 或任何产生BUG的文件
```

然后你会比gdb反汇编更清楚的知道发生了什么。

现在，问题是把你所拥有的所有数据结合起来：C源码（关于它应该怎样的一般知识），汇编代码及其反汇编得到的代码（另外还有从“oops”消息得到的寄存器状态-对了解毁坏

的
指针有用，而且当你有了汇编代码你也能拿其它的寄存器和任何它们对应的C表达式做匹配）。

实际上，你仅需看看哪里不匹配（这个例子是“Code”反汇编和编译器生成的代码不匹配）。
然后你须要找出为什么不匹配。通常很简单-你看到代码使用了空指针然后你看代码想知道空指针是怎么出现的，还有检查它是否合法..

现在，如果明白这是一项耗时的工作而且需要一丁点儿的专心，没错。这就是我为什么大多只是忽略那些没有符号表信息的崩溃报告的原因：简单的说太难查找了（我有一些
程序用于在内核代码段中搜索特定的模式，而且有时我也已经能找出那些崩溃的地方，但是仅仅是找出正确的序列也确实需要相当扎实的内核知识）

_有时_会发生这种情况，我仅看到崩溃中的反汇编代码序列， 然后我马上就明白问题出在哪里。这时我才意识到自己干这个工作已经太长时间了;-)

Linus

关于Oops跟踪的注解：

为了帮助Linus和其它内核开发者，klogd纳入了大量的支持来处理保护错误。为了拥有对地址解析的完整支持至少应该使用1.3-pl3的sysklogd包。

当保护错误发生时，klogd守护进程自动把内核日志信息中的重要地址翻译成它们相应的符号。

klogd执行两种类型的地址解析。首先是静态翻译其次是动态翻译。静态翻译和ksymoops一样使用System.map文件。为了做静态翻译klogd守护进程必须在初始化时能找到system map文件。关于klogd怎样搜索map文件请参看klogd手册页。

动态地址翻译在使用内核可装载模块时很重要。 因为内核模块的内存是从内核动态内存池里分配的，所以不管是模块开始位置还是模块中函数和符号的位置都不是固定的。

内核支持允许程序决定装载哪些模块和它们在内存中位置的系统调用。使用这些系统调用klogd守护进程生成一张符号表用于调试发生在可装载模块中的保护错误。

至少klogd会提供产生保护错误的模块名。还可有额外的符号信息供可装载模块开发者选择以从模块中输出符号信息。

因为内核模块环境可能是动态的，所以必须有一种机制当模块环境发生改变时来通知klogd守护进程。 有一些可用的命令行选项允许klogd向当前执行中的守护进程发送信号，告知符号信息应该被刷新了。 更多信息请参看klogd手册页。

sysklogd发布时包含一个补丁修改了modules-2.0.0包，无论何时一个模块装载或者卸载都会自动向klogd发送信号。打上这个补丁提供了必要的对调试发生于内核可装载模块的保护错误的无缝支持。

以下是被klogd处理过的发生在可装载模块中的一个保护错误例子：

```
Aug 29 09:51:01 blizard kernel: Unable to handle kernel paging request at
virtual address f15e97cc
Aug 29 09:51:01 blizard kernel: current->tss.cr3 = 0062d000, %cr3 = 0062d000
Aug 29 09:51:01 blizard kernel: *pde = 00000000
```

```
Aug 29 09:51:01 blizard kernel: Oops: 0002
Aug 29 09:51:01 blizard kernel: CPU:    0
Aug 29 09:51:01 blizard kernel: EIP:    0010:[oops:_oops+16/3868]
Aug 29 09:51:01 blizard kernel: EFLAGS: 00010212
Aug 29 09:51:01 blizard kernel: eax: 315e97cc   ebx: 003a6f80   ecx: 001be77b
edx: 00237c0c
Aug 29 09:51:01 blizard kernel: esi: 00000000   edi: bffffdb3   ebp: 00589f90
esp: 00589f8c
Aug 29 09:51:01 blizard kernel: ds: 0018   es: 0018   fs: 002b   gs: 002b   ss:
0018
Aug 29 09:51:01 blizard kernel: Process oops_test (pid: 3374, process nr: 21,
stackpage=00589000)
Aug 29 09:51:01 blizard kernel: Stack: 315e97cc 00589f98 0100b0b4 bffffed4
0012e38e 00240c64 003a6f80 00000001
Aug 29 09:51:01 blizard kernel:        00000000 00237810 bfffff00 0010a7fa
00000003 00000001 00000000 bfffff00
Aug 29 09:51:01 blizard kernel:        bffffdb3 bffffed4 ffffffda 0000002b
0007002b 0000002b 0000002b 00000036
Aug 29 09:51:01 blizard kernel: Call Trace: [oops:_oops_ioctl+48/80]
[_sys_ioctl+254/272] [_system_call+82/128]
Aug 29 09:51:01 blizard kernel: Code: c7 00 05 00 00 00 eb 08 90 90 90 90 90 90
90 90 89 ec 5d c3
```

Dr. G.W. Wettstein   Oncology Research Div. Computing Facility
Roger Maris Cancer Center   INTERNET: greg@wind.rmcc.com
820 4th St. N.
Fargo, ND 58122
Phone: 701-234-7556

受污染的内核

一些oops报告在程序记数器之后包含字符串'Tainted: '。这表明内核已经被一些东西给污染了。 该字符串之后紧跟着一系列的位置敏感的字符，每个代表一个特定的污染值。

1：'G'如果所有装载的模块都有GPL或相容的许可证，'P'如果装载了任何的专有模块。没有模块MODULE_LICENSE或者带有insmod认为是与GPL不相容的的MODULE_LICENSE的模块被认定是专有的。

2：'F'如果有任何通过“insmod -f”被强制装载的模块，' '如果所有模块都被正常装载。

3：'S'如果oops发生在SMP内核中，运行于没有证明安全运行多处理器的硬件。 当前这种情况仅限于几种不支持SMP的速龙处理器。

4：'R'如果模块通过“insmod -f”被强制装载，' '如果所有模块都被正常装载。

5：'M'如果任何处理器报告了机器检查异常，' '如果没有发生机器检查异常。

6：'B'如果页释放函数发现了一个错误的页引用或者一些非预期的页标志。

7：'U'如果用户或者用户应用程序特别请求设置污染标志，否则' '。

8：'D'如果内核刚刚死掉，比如有OOPS或者BUG。

使用'Tainted: '字符串的主要原因是要告诉内核调试者，这是否是一个干净的内核亦或发生了任何的不正常的事。污染是永久的：即使出错的模块已经被卸载了，污染值仍然存在，以表明内核不再值得信任。