**主要内容：**

* 内核调试的难点
* 内核调试的工具和方法
* 总结

### ****1. 内核调试的难点****

内核调试的难点大致有以下几个：

1. 重现bug困难 - 如果能够重现一个bug, 相当于成功了一半. (特别是有些bug和硬件相关, 执行几百万次之后才有一次错误)
2. 调试风险比较大 - 稍有不慎, 即造成系统崩溃
3. 定位bug的初始版本困难 - 内核版本更新很快, 很难确定bug是在那个版本开始出现的

### ****2. 内核调试的工具和方法****

内核调试虽然困难, 但同时也极具挑战性, 如果能够解决一个困扰大家多时的内核bug, 那将会给自己带来极大的成就感. :)

而且, 随着内核的不断发展, 内核调试的手段和方法也在不断进步, 下面是书中提到的一些常用的调试手段.

#### **2.1 输出 LOG**

输出LOG不光是内核调试, 即使是在用户态程序的调试中, 也是经常使用的一个调试手段.

通过在可疑的代码周围加上一些LOG输出, 可以准确的了解bug发生前后的一些重要信息.

##### **2.1.1 LOG等级**

linux内核中输出LOG的函数是 printk (语法和printf几乎雷同, 唯一的区别是printk可以指定日志级别)

printk之所以好用, 就在与它随时都可以被调用, 没有任何限制条件.

printk的输出日志级别如下:

|  |  |
| --- | --- |
| **等级** | **描述** |
| KERN\_EMERG | 一个紧急情况 |
| KERN\_ALERT | 一个需要立即被注意到的错误 |
| KERN\_CRIT | 一个临界情况 |
| KERN\_ERR | 一个错误 |
| KERN\_WARNING | 一个警告 |
| KERN\_NOTICE | 一个普通的, 不过也有可能需要注意的情况 |
| KERN\_INFO | 一条非正式的消息 |
| KERN\_DEBUG | 一条调试信息--一般是冗余信息 |

输出示例:

printk(KERN\_WARNING "This is a warning!\n");

printk(KERN\_DEBUG "This is a debug notice!\n");

##### **2.1.2 LOG记录**

标准linux系统上, printk 输出log之后, 由用户空间的守护进程klogd从缓冲区中读取内核消息, 然后再通过syslogd守护进程将它们保存在系统日志文件中.

syslogd 将接受到的所有内核消息添加到一个文件中, 该文件默认为: /var/log/dmesg (系统Centos6.4 x86\_64)

PS. 上篇博客中的内核模块的输出LOG, 都可以在 /var/log/dmesg 中看到

#### **2.2 oops**

oopss是个拟声词, 类似 "哎哟" 的意思. 它是内核通知用户有不幸发生的最常用方式.

触发一个oops很简单, 其实只要在上篇博客中的那些内核模块示例中随便找一个, 里面加上一段给未初始化的指针赋值的代码, 就能触发一个oops

oops中包含错误发生时的一些重要信息(比如, 寄存器上下文和回溯线索), 对调试bug很有帮助!

调试内核时, 还可以开启内核编译参数中的各种和内核调试相关的选项, 那样还可以给我们提供内核崩溃时的一些额外信息.

#### **2.3 主动触发bug**

调试中有时将某些情况下标记为bug, 执行到这些情况时, 提供断言并输出信息.

BUG 和 BUG\_ON 就是2个可以主动触发oops的内核调用.

在不应该被执行到的地方使用 BUG 或者 BUG\_ON 来捕获.

比如:

if (bad\_thing)

BUG();// 或者

BUG\_ON(bad\_thing);

如果想要触发更为严重的错误, 可以使用 panic() 函数

比如:

if (terrible\_thing)

panic("terrible thing is %ld\n", terrible\_thing);

此外, 还有dump\_stack 函数可以打印寄存器上下文和回溯信息.

比如:

if (!debug\_check) {

printk(KERN\_DEBUG "provide some information...\n");

dump\_statck();

}

#### **2.4 神奇的系统请求键**

这个系统请求键之所以神奇, 在于它可以在一个快挂了的系统上输出一些有用的信息.

这个按键一般就是标准键盘上的 [SysRq] 键 (就在 F12 键右边, 其实就是windows中截整个屏幕的按键)

单独按那个键相当于截屏, 按住 ALT + [SysRq] = [SysRq]的功能

启用这个键的功能有2个方法:

* 开启内核编译选项 : CONFIG\_MAGIC\_SYSRQ
* 动态启用: echo 1 > /proc/sys/kernel/sysrq

支持 SysRq 的命令如下: (注意要在控制台界面下使用这个键, 比如通过 ALT+CTRL+F2 进入一个控制台界面)

|  |  |
| --- | --- |
| **主要命令** | **描述** |
| SysRq-b | 重新启动机器 |
| SysRq-e | 向init以外的所有进程发送SIGTERM信号 |
| SysRq-h | 在控制台显示SysRq的帮助信息 |
| SysRq-i | 向init以外的所有进程发送SIGKILL信号 |
| SysRq-k | 安全访问键:杀死这个控制台上的所有程序 |
| SysRq-l | 向包括init的所有进程发送SIGKILL信号 |
| SysRq-m | 把内存信息输出到控制台 |
| SysRq-o | 关闭机器 |
| SysRq-p | 把寄存器信息输出到控制台 |
| SysRq-r | 关闭键盘原始模式 |
| SysRq-s | 把所有已安装文件系统都刷新到磁盘 |
| SysRq-t | 把任务信息输出到控制台 |
| SysRq-u | 卸载所有已加载文件系统 |

#### **2.5 内核调试器 gdb和kgdb**

linux内核的调试器可以使用 gdb或者kgdb, 配置比较麻烦, 准备实际用调试的时候再去试试效果如何..

#### **2.6 探测系统**

下面一些方法是在修改内核后, 用来试探内核反应的小技巧.

##### **2.6.1 用UID控制内核执行**

比如在内核中加入了新的特性, 为了测试特性, 可以用UID来控制内核是否执行新特性.

if (current->uid != 7777) {

/\* 原先的代码 \*/

} else {

/\* 新的特性 \*/

}

##### **2.6.2 用条件变量控制内核执行**

也可以设置一些条件变量来控制内核是否执行某段代码.

条件变量可以像上篇博客中那样, 设置在 sys 文件系统的某个文件中. 当文件中的值变化时, 通知内核执行相应的代码.

##### **2.6.3 使用统计量观察内核执行某段代码的频率**

实现思路就是在内核中的设置一个全局变量, 比如 my\_count, 当内核执行到某段代码时, 给 my\_count + 1 就行.

同时还要将 my\_count 打印出来(可以用printk), 便于随时查看它的值.

##### **2.6.4 控制内核执行某段代码的频率**

有时侯, 我们需要在内核发生错误时打印错误相关的信息, 如果这个错误不会导致内核崩溃, 并且这个错误每秒会发生几百次甚至更多.

那么, 用printk输出的信息会非常多, 给系统造成额外的负担.

这时, 我们就需要想办法控制错误输出的频率, 有2种方法:

**方法1:** 隔一段时间才输出一次错误

static unsigned long prev\_jiffy = jiffies; /\* 频率限制 \*/

if (time\_after(jiffies, prev\_jiffy + 2\*HZ)) { /\* 输出间隔至少 2HZ \*/

prev\_jiffy = jiffies;

printk(KERN\_ERR "错误信息....\n");

}

**方法2:** 输出 N 次之后, 不再输出(N是正整数)

static unsigned long limit = 0;

if (limit < 5) { /\* 输出5次错误信息后就不再输出 \*/

limit++;

printk(KERN\_ERR "错误信息....\n");

}

#### **2.7 二分法查找bug发生的最初内核版本**

在内核发生了bug之后, 如果能够知道是bug从哪个内核版本开始出现的, 那对修正这个bug会有很大的帮助.

由于内核代码非常庞大, 即使用二分查找法, 手工去找哪个版本开始出现bug的话, 仍然是非常耗时和繁琐的.

好在 Git 给我们提供了一个非常有用的二分搜索机制.

git bisect start # 开始二分搜索

git bisect bad <bad\_revision> # 指定一个bug出现的内核版本号

git bisect good <good\_revision> # 指定一个没有bug的内核版本号, 此时git会检测2个版本直接的隐患

# 根据结果再次设置 bad 和 good 的版本号, 缩小Git检索范围, 直至找到可疑之处为止.

#### **2.8 社区**

当你在调试bug时用尽了一切手段仍然无济于事时, 可以考虑求助linux社区, 求助时注意一定要描述清楚bug的状况.

(可以参考一下别人汇报bug的格式)

linux 内核相关的邮件列表非常多: 参见 <http://vger.kernel.org/vger-lists.html>

和内核开发, bug汇报相关的邮件列表参见: [http://vger.kernel.org/vger-lists.html#linux-kernel](http://vger.kernel.org/vger-lists.html" \l "linux-kernel) (这个邮件列表非常活跃, 每天的邮件都非常多!!!)

### ****3. 总结****

linux内核调试必须要依靠大量的实践来掌握, 仅仅靠上面介绍的一些技巧还远远不够, 只有实实在在的去阅读内核代码, 实实在在的去修正一个个bug, 才算真正掌握内核, 真正了解内核.

多看看之前linux内核bug的修正案例, 也是个不错的积累经验的方法.

PS.

对于初学者来说, 在真机上做内核开发动辄导致机器崩溃(panic), 非常麻烦. 现在的虚拟机这么强大, 建议都在虚拟机上测试linux内核修改的效果.

我之前的关于<<Linux内核设计与实现>>笔记的博客中的代码都是在虚拟机上运行测试的.