9 minutes

Linux 内核调试方法

Table of Contents

> 基于 Ubuntu 14.04 ，Linux Kernel 4.0 以上版本。

1. printk()

`printk()` 是内核提供的函数，用于将内核空间的信息打印到用户空间缓冲区，打印的信息可以通过 demsg 命令查看，或者直接查看 /proc/kmsg 文件。缓冲区是一个环形队列的结构，消息太多时，旧的消息就会被逐渐覆盖，缓冲区大小是在 kernel/printk/printk.c 文件中的代码设置的：

```
#define __LOG_BUF_LEN (1 << CONFIG_LOG_BUF_SHIFT)
static char __log_buf[__LOG_BUF_LEN] __aligned(LOG_ALIGN);
```

缓冲区大小是 CONFIG_LOG_BUF_SHIFT*2 个字节，CONFIG_LOG_BUF_SHIFT 是在 init/Kconfig 文件中设置的，我们可以在 menuconfig 的相关路径中修改：

```
General setup -> Kernel log buffer size(16 => 64KB, 17 => 128kB)
```

还可以在加载内核时用启动参数 log_buf_len=n[KMG] 设置，其中的 n 必须是 2 的整数倍。

在调用 `printk()` 函数时要设置消息级别，从 0 到 7 ，数值越小级别越高，相应的宏定义在 include/linux/kern_levels.h 文件中：

```
#define KERN_EMERG      KERN_SOH "0"    /* system is unusable */
#define KERN_ALERT      KERN_SOH "1"    /* action must be taken immediatel
#define KERN_CRIT       KERN_SOH "2"    /* critical conditions */
#define KERN_ERR        KERN_SOH "3"    /* error conditions */
#define KERN_WARNING    KERN_SOH "4"    /* warning conditions */
#define KERN_NOTICE     KERN_SOH "5"    /* normal but significant conditio
#define KERN_INFO       KERN_SOH "6"    /* informational */
#define KERN_DEBUG      KERN_SOH "7"    /* debug-level messages */

#define KERN_DEFAULT    KERN_SOH "d"    /* the default kernel loglevel */
```

内核中还有一个默认日志级别，只有数值小于这个级别的消息才会被打印到控制台上，大于或者等于这个数值的消息不会显示，它设置在 lib/Kconfig.debug 文件中，缺省情况下会设为 KERN_WARNING(4)，我们可以在 menuconfig 的相关路径中设置：

```
Kernel hacking -> printk and dmesg options -> Default message log level(1-
```

也可以用内核启动参数 loglevel=n 设置，n 的取值是 0~7 。如果直接设置了启动参数 debug，那么日志级别就是 KERN_DEBUG(7)，所有调试信息都会显示在控制台上。还可以在系统启动后，在 /proc/sys/kernel/printk 文件中调整 `printk()` 函数的输出等级，该文件有四个数值，各自的含义：

1. 控制台的日志级别:当前的打印级别,优先级高于该值(值越小,优先级越高)的消息将被打印至控制台
2. 默认的消息日志级别: 将用该优先级来打印没有优先级前缀的消息,也就是直接写 `printk("xxx")` 而不带打印级别的情况下,会使用该打印级别
3. 最低的控制台日志级别: 控制台日志级别可被设置的最小值(一般是1)
4. 默认的控制台日志级别: 控制台日志级别的默认值

修改方法：

```
root@sh-VirtualBox:/proc/sys/kernel# cat printk
4   4   1   7
root@sh-VirtualBox:/proc/sys/kernel# echo 5 > printk
root@sh-VirtualBox:/proc/sys/kernel# cat printk
5   4   1   7
root@sh-VirtualBox:/proc/sys/kernel# echo  "5 5" > printk
root@sh-VirtualBox:/proc/sys/kernel# cat printk
5   5   1   7
```

默认情况下， `printk()` 打印的消息是带时间戳的，可以在 menuconfig 的相应路径下关闭或者打开：

```
Kernel hacking -> printk and dmesg options -> Show timing information on p
```

为了方便调用，内核提供很多封装了 `printk()` 函数的宏，在 `/include/linux/printk.h` 头文件中声明的 `pr_xxx()` ，例如：

```
#define pr_fmt(fmt) fmt
#define pr_err(fmt, ...) printk(KERN_ERR pr_fmt(fmt), ##__VA_ARGS__)
#define pr_warning(fmt, ...) printk(KERN_WARNING pr_fmt(fmt), ##__VA_ARGS_
#define pr_info(fmt, ...) printk(KERN_INFO pr_fmt(fmt), ##__VA_ARGS__)
```

我们可用通过 `pr_fmt(fmt)` 添加一些自定义的消息格式，例如：

```
#define pr_fmt(fmt) "[driver] watchdog:" fmt
```

这里要注意 `pr_debug()` ，它与其他的宏不同，需要满足如下两个条件之一才会打印信息：

1. 在源文件、或者编译时定义了 DEBUG 宏，这个方式在开发内核模块时很有用
2. 开启了 CONFIG_DYNAMIC_DEBUG ，也就是 menuconfig 中的 `Kernel hacking -> printk and dmesg options`

这里还有一个问题，内核启动后，需要一段时间才能准备好控制台，这段时间内的内核信息是无法通过控制台显示，内核为此提供了 early printk 机制，它会在内核启动后就注册一个 boot console ，让后将内核信息显示在这个控制台上。使能 early printk 的方法有两步：

1. 在 menuconfig 中打开 Early printk： `Kernel hacking -> Early printk`
2. 在启动参数中设置 earlyprintk=[vga|serial][,ttySn[,baudrate]][,keep]

如果用户空间的 `printf()` 和内核空间的 `printk()` 同时执行，二者的输出会互相干扰，内核为此提供了 /dev/ttyprintk 设备文件，可以将用户空间的信息打印到这个设备中，这样用户信息与内核信息就会顺序输出，输出的消息会自带 [U] 前缀。对于没有 /dev/ttyprintk 设备的系统，可以用 /dev/kmsg 代替，只是没有了 [U] 标识，需要用户自己添加前缀。

2. SysRq 键

标准键盘的右上角有一个 PrintScreen/SysRq 键，它的一个功能是截屏，另一个功能是当系统死机无法输入命令时，用这个按键获取内核信息。SysRq 键在确认内核运行、调查死机原因等情况时非常有效。关于它的详细情况可以参考内核的 Documentation/sysrq.txt 文件。

要使用 SysRq 键，需要启动内核配置 CONFIG_MAGIC_SYSRQ ，在 menuconfig 中的路径是：

```
Kernel hacking -> Magic SysRq key
```

系统启动后，就可以在 /proc/sys/kernel/sysrq 文件中设置 SysRq 按键的功能，该文件的默认值是内核选项 CONFIG_MAGIC_SYSRQ_DEFAULT_ENABLE 设置的,必须是十六进制，在 menuconfig 的路径是：

```
Kernel hacking -> (0x01) Enable magic Sysrq key functions by default
```

> 注意，/proc/sys/kernel/sysrq 设置的各项功能，只对从键盘和串口控制台的输入有效，对于远程 ssh 等方式无效。直接向 /proc/sysrq-

> trigger 写入命令键则不受限制： `echo [command key] > /proc/sysrq-trigger` 。

这个文件的值是位掩码，取值如下，括号内是命令键：

- 0 ，禁用 sysrq
- 1 ，使能所有 sysrq 功能
- 2 = 0x2 ，允许控制控制台日志级别（0~9）
- 4 = 0x4 ，使能键盘控制 (kr)
- 8 = 0x8 ，使能显示进行等信息（lptwmcz）
- 16 = 0x10 ，使能 sync 命令（s）
- 32 = 0x20 ，使能只读状态下的重新挂在（u）
- 64 = 0x40 ，使能进程信号，例如 term, kill（ei）
- 128 = 0x80 ，使能重启和关机（b）
- 256 = 0x100 ，允许控制实时任务（q）

可以直接修改这个文件的值，比如使能 sync 和重新挂载：

```
# echo 48 > /proc/sys/kernel/sysrq
```

也可以在 `/etc/sysctl.d/10-magic-sysrq.conf` 文件中修改 kernel.sysrq 选项（也可能在 `/etc/sysctl.conf` 文件中）。配置好功能后，通过组合键 `Alt-SysRq-<command key>` 就可以使用 SysRq 键的各项功能，功能键如下：

```
'b' - Will immediately reboot the system without syncing or unmounting you
'c' - Will perform a system crash by a NULL pointer dereference. A crashdu
'd' - Shows all locks that are held.
'e' - Send a SIGTERM to all processes, except for init.
'f' - Will call the oom killer to kill a memory hog process, but do not pa
'g' - Used by kgdb (kernel debugger)
'h' - Will display help (actually any other key than those listed here wil
'i' - Send a SIGKILL to all processes, except for init.
'j' - Forcibly "Just thaw it" - filesystems frozen by the FIFREEZE ioctl.
'k' - Secure Access Key (SAK) Kills all programs on the current virtual co
'l' - Shows a stack backtrace for all active CPUs.
'm' - Will dump current memory info to your console.
'n' - Used to make RT tasks nice-able
'o' - Will shut your system off (if configured and supported).
'p' - Will dump the current registers and flags to your console.
'q' - Will dump per CPU lists of all armed hrtimers (but NOT regular timer
'r' - Turns off keyboard raw mode and sets it to XLATE.
's' - Will attempt to sync all mounted filesystems.
't' - Will dump a list of current tasks and their information to your cons
'u' - Will attempt to remount all mounted filesystems read-only.
'v' - Forcefully restores framebuffer console
'v' - Causes ETM buffer dump [ARM-specific]
'w' - Dumps tasks that are in uninterruptable (blocked) state.
'x' - Used by xmon interface on ppc/powerpc platforms. Show global PMU Reg
'y' - Show global CPU Registers [SPARC-64 specific]
'z' - Dump the ftrace buffer
'0'-'9' - Sets the console log level, controlling which kernel messages wi
```

如果系统疑似死机，可以一次执行 s-u-b 命令重启内核，如果不需要重启，可以执行 c 命令提取崩溃转储，获取内核信息（内核崩溃转储是指将系统内存的内容输出到文件）。还可以尝试用 i 命令向进程发送 SIGKILL 信号，使系统恢复。

3. Kdump

Kdump 是内核提供的崩溃转储功能，工作原理是在系统内核崩溃时启动一个特殊的 dump-capture kernel 把系统内存里的数据保存到磁盘文件中，由内核机制和用户空间工具共同完成。Dump-capture kernel 可以是独立的，也可以和系统内核集成在一起（这需要硬件支持）。Kdump 的工作过程如下：

1. 系统内核启动的时候，要给 dump-capture kernel 预留一块内存空间；
2. 内核启动完成后，用户空间的 kdump service 执行 `kexec -p` 命令把 dump-capture kernel 载入预留的内存里（`/sys/kernel/kexec_crash_loaded` 的值为 1 表示已经加载）；
3. 如果系统发生 crash，生产内核会自动 reboot 进入 dump-capture kernel，dump-capture kernel 只使用自己的预留内存，确保其余的内存数据不会被改动，它的任务是把系统内存里的数据写入到 dump 文件，比如 /var/crash/vmcore，为了减小文件的大小，它会通过 makedumpfile(8) 命令对内存数据进行挑选和压缩；
4. dump 文件写完之后，dump-capture kernel 自动 reboot 。

预留内存的方法是用内核启动参数 `crashkernel=size[@offset]` 实现的，某些内核支持 `crashkernel=auto` 自动分配大小，如果不支持，或者系统没有足够内存，就需要手动设置。通常 offset 可以设置为 16MB(0x1000000)，size 根据系统内存的大小设置，而且要与 64MB 对齐：

1. 如果系统内存小于 512MB ，则不要保留内存
2. 如果系统内存介于 512MB 到 2GB 之间，可以保留 64MB 内存
3. 如果系统内存大于 2GB ，可以保留 128MB 内存

可能导致内核崩溃的事件包括：

- Kernel Panic
- Non Maskable Interrupts (NMI)
- Machine Check Exceptions (MCE)
- Hardware failure
- Manual intervention

对于某些崩溃事件（例如 panic、NMI），内核会自动做出反应，并通过 kexec 触发崩溃转储，其他情况下需要手动捕获内存信息。

在 Ubuntu 上首先要安装内核崩溃转储工具：

```
$ sudo apt-get install linux-crashdump
```

> 如果是 Fedora 操作系统，通常是安装 crash 和 kexec-tools 软件包。

linux-crashdump 包安装了三个工具，分别是：crash，kexec-tools 和 makedumpfile。安装过程中会出现如下对话框，选择 Yes ，表示默认使能 kdump ：

```
|-----------------------| Configuring kdump-tools |----------------------
|
|
```

```
| If you choose this option, the kdump-tools mechanism will be enabled. A
| reboot is still required in order to enable the crashkernel kernel
| parameter.
|
| Should kdump-tools be enabled by default?
|
|                    <Yes>                        <No>
|
|-------------------------------------------------------------------------
```

然后编辑 `/etc/default/kdump-tools` 文件，修改选项 `USE_KDUMP=1` ，使能内核加载 kdump ，然后重启系统，内核自动激活 crashkernel= 启动参数 ，kdump-tools 默认启动，用 `kdump-config show` 命令和 `/sys/kernel/kexec_crash_loaded` 文件查看 kdump 的配置和状态，在 `/proc/cmdline` 文件中查看 crashkernel 的设置：

```
$ kdump-config show
DUMP_MODE:        kdump
USE_KDUMP:        1
KDUMP_SYSCTL:     kernel.panic_on_oops=1
KDUMP_COREDIR:    /var/crash
crashkernel addr: 0x2d000000
current state:    ready to kdump

kexec command:
/sbin/kexec -p --command-line="BOOT_IMAGE=/boot/vmlinuz-4.4.0-31-generic r
$ cat /sys/kernel/kexec_crash_loaded
1
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.4.0-31-generic root=UUID=2744d8e0-18c2-493f-b61
```

系统启动后，可以通过向 `/sys/kernel/kexec_crash_size` 写入一个比原来小的数值来缩小甚至完全释放 crashkernel 。然后执行 `sudo kdump-config load` 加载 kdump ，也可以把 /etc/init.d/kdump-tool 服务设为默认启动，这样系统会自动加载。准备工作完成后，尝试提取崩溃转储，先确保 sysrq=1 ，然后手动触发一次崩溃：

```
# echo c > /proc/sysrq-tirgger
```

稍等片刻，如果转储成功，内核会自动重启，并且在 /var/crash/ 目录下生成转储文件：

```
$ ls -l /var/crash/*
total 28
drwxr-sr-x 2 root whoopsie  4096  7月  6 11:45 201807061145
-rw-r----- 1 root whoopsie 18095  7月  6 11:45 linux-image-4.4.0-31-generi
$ ls -l /var/crash/201807061145/
total 55300
-rw------- 1 root whoopsie    41223  7月  6 11:45 dmesg.201807061145
-rw------- 1 root whoopsie 56578589  7月  6 11:45 dump.201807061145
```

> 转储需要时间，如果手动关机重启会导致转储不完整，数据无法解读。

如果是 RedHat 系统，生成的转储文件是 vmcore ，可以直接用 crash 命令分析。而 Ubuntu 提供了叫做 Apport 的工具，将系统内其他有用的信息一起打包生成了 linux-image-4.4.0-31-generic-201807061145.crash 文件，而以时间戳命名的文件夹 201807061145 包含了 dmesg 信息文件和 kdump 转储文件，对于某些版本，这两个文件也包含在 crash 文件中。对 crash 文件解压后可以得到几个与系统信息有关的纯文本文件：

```
$ sudo apport-unpack /var/crash/linux-image-4.4.0-31-generic-201807061145.
$ ls ~/201807061145.crash
Architecture  Date  DistroRelease  Package  ProblemType  Uname  VmCoreDmes
```

4. 崩溃测试

内核有一个 lkdtm 模块，Linux Kernel Dump Test Module ，通过各种方式使内核崩溃，用于测试崩溃转储的功能。通常发行版的内核不会使能这个模块，需要启用内核 CONFIG_LKDTM 选项，在 menuconfig 的路径是：

```
Kernel hacking -> RunTime Testing -> Linux Kernel Dump Test Tool Module
```

最好编译成模块，然后加载模块时，通过模块参数指定崩溃位置和崩溃原因，即可造成所需的内核崩溃。

5. crash 命令

crash 是一个强大的交互式工具，基于 gdb ，用于分析内核映像，比如内核崩溃转储信息。有些系统中，安装 linux-crashdump 时会包含 crash ，如果没有，需要手动安装：

```
sudo apt-get install crash
```

分析之前需要安装系带有 debug-info 的内核，叫做 kernel-debuginfo ，这是 redhat 的叫法， ubuntu 下叫 debug symbols，简称 dbgsym 。 ubuntu 默认安装时不会安装 dbgsym ，默认仓库上也没有 dbgsym 包。 dbgsym 包存在于独立的仓库上，官方仓库地址为 `http://ddebs.ubuntu.com/` ，安装方法参考：https://oolap.com/2015-11-07-ubuntu-install-dbgsym 。kernel-debuginfo 的版本应该和系统运行的内核版本完全一致，如果是自行编译的内核，可能无法在官方仓库中找到对应版本的 kernel-debuginfo ，这时可以自行编译安装 kernel-debuginfo ，参考下一节。安装完成后，会在 `/usr/lib/debug/boot/` 目录下生成带有调试信息的 vmlinux ，然后用 crash 工具分析 kdump 生成崩溃转储信息：

```
$ sudo crash  /usr/lib/debug/boot/vmlinux-4.4.0-31-generic /var/crash/2018
```

下面以一个 Fedora14(kernel 2.6.37) 下产生的转储文件 vmcore 为例说明 crash 的用法，crash 成功启动后先打印一段转储文件的分析报告，包括崩溃时间、崩溃类型、CPU、内存等，然后进入一个交互环境：

```
KERNEL: /boot/vmlinux
DUMPFILE: vmcore
CPUS: 1
DATE: Fri Jul 27 13:59:13 2018
UPTIME: 00:05:23
LOAD AVERAGE: 0.01, 0.11, 0.07
TASKS: 56
NODENAME: localhost.localdomain
RELEASE: 2.6.37.6
VERSION: #11 SMP Thu Jul 26 15:42:06 CST 2018
MACHINE: i686 (1500 Mhz)
MEMORY: 1 GB
PANIC: "[  323.903003] Oops: 0002 [#1] SMP " (check log for details)
PID: 4437
COMMAND: "bash"
TASK: f6ec0c90 [THREAD_INFO: f6d2e000]
CPU: 0
STATE: TASK_RUNNING (PANIC)
crash >
```

可以看到引起崩溃的进程是 `PID: 4437` ，crash 提供了 ps 命令显示所有进程的状态，用 `ps | grep 4437` 可以筛选出引起崩溃的进程：

```
crash > ps | grep 4437
  PID    PPID    CPU       TASK      ST   %MEM    VSZ    RSS   COMM
  4437   4426    0    f6ec0c90       RU    0.2   8064   1780   bash
```

bt 命令用于输出某个进程的内核栈的遍历，没有指定 PID 时默认输出引起崩溃的进程的内核栈信息：

```
crash> bt
PID: 4437   TASK: f6ec0c90  CPU: 0   COMMAND: "bash"
#0 [f6d2fdec] crash_kexec at c0466264
#1 [f6d2fe2c] __bad_area_nosemaphore at c04225b5
#2 [f6d2fe48] bad_area at c042260c
#3 [f6d2fe60] do_page_fault at c079c8c9
#4 [f6d2fed8] error_code (via page_fault) at c079a685
EAX: 00000063  EBX: 00000063  ECX: ffffffd6  EDX: 00000000  EBP: f6d2ff18
DS:  007b      ESI: c095dfe0  ES:  007b      EDI: 00000004  GS:  00e0
CS:  0060      EIP: c061e0b9  ERR: ffffffff  EFLAGS: 00010046
#5 [f6d2ff0c] sysrq_handle_crash at c061e0b9
#6 [f6d2ff1c] __handle_sysrq at c061e63d
#7 [f6d2ff40] write_sysrq_trigger at c061e6e2
#8 [f6d2ff50] proc_reg_write at c0507c84
#9 [f6d2ff74] vfs_write at c04cdf4c
#10 [f6d2ff90] sys_write at c04ce11d
#11 [f6d2ffb0] ia32_sysenter_target at c0403298
EAX: 00000004  EBX: 00000001  ECX: b77f8000  EDX: 00000002
DS:  007b      ESI: b77f8000  ES:  007b      EDI: 00000002
SS:  007b      ESP: bfc32fd0  EBP: bfc33008  GS:  0033
CS:  0073      EIP: b77fc424  ERR: 00000004  EFLAGS: 00000246
```

可以看到系统崩溃前最后一条调用是 `#5 [f6d2ff0c] sysrq_handle_crash at c061e0b9` ，我们用 `dis` 命令看一下这个地址的反汇编结果：

```
crash> dis -l c061e0b9
/usr/src/linux-2.6.37/drivers/tty/sysrq.c: 134
0xc061e0b9 <sysrq_handle_crash+23>:     movb   $0x1,0x0
```

出错的代码位于 `/usr/src/linux-2.6.37/drivers/tty/sysrq.c` 文件的 134 行：

```
129 static void sysrq_handle_crash(int key)
130 {
131     char *killer = NULL;
132     panic_on_oops = 1;      /* force panic */
133     wmb();
134     *killer = 1;
135 }
```

这里为指针赋值 `*killer = 1` ，而 131 行定义的是一个空指针，比如出错。

crash 还有很多命令：

- log： 打印系统消息缓冲区，从而可能找到系统崩溃的线索。
- sys： 显示系统概况。
- kmem： 显示内存使用信息。
- irq： 显示中断的信息。
- mod： 显示模块信息。
- runq： 显示处于运行队列的进程。
- struct： 显示结构的定义、地址和数据。

6. kernel-debuginfo

kernel-debuginfo 是指带有 Debug information 的内核，就是在编译内核是指定 CONFIG_DEBUG_INFO 等相关选项，在 menuconfig 的路径是：

```
Kernel hacking -> Kernel debugging -> Compile the kernel with debug info
```

与 Kdump 分析相关的选项还有：

- kexec system call： CONFIG_KEXEC=y
- sysfs file system support： CONFIG_SYSFS=y
- Compile the kernel with debug info： CONFIG_DEBUG_INFO=Y
- kernel crash dumps： CONFIG_CRASH_DUMP=y
- /proc/vmcore support： CONFIG_PROC_VMCORE=y

编译成功后，就会在源码目录下生成带有 debuginfo 的内核镜像 vmlinux ，kdump 、crash 等内核调试方法都会用到它。vmlinux 是一个包含 Linux kernel 的静态链接的可执行文件，ELF 格式。通常 /boot 目录下启动的内核是 vmlinuz ，它是 vmlinux 经过 gzip 和 objcopy 制作出来的压缩文件。vmlinuz 是一种统称，有两种具体的表现形式 zImage 和 bzImage 。bzimage 和 zImage 的区别在于本身的大小，以及加载到内存时的地址不同，zImage在 0～640KB，而bzImage 则在 1M 以上的位置。

不同的程序查找这个内核的路径是不一样的，通常需要在如下路径建立这个内核的符号链接：

```
/boot/vmlinux-`uname -r`
/usr/lib/debug/lib/modules/`uname -r`/vmlinux
/lib/modules/`uname -r`/vmlinux
```

有些程序还需要在 `/lib/modules/` 目录下建立内核源码树和构建目录的符号链接：

```
/lib/modules/`uname -r`/source
/lib/modules/`uname -r`/build
```

7. NMI

NMI(non-maskable interrupt) 就是不可屏蔽的中断，当 x86 发生了无法恢复的硬件故障后，会触发这个中断通知操作系统，如果操作系统配置了 kdump，还会触发崩溃转储。根据 Intel 的软件开发者手册第三卷 6.7 的描述，NMI 的来源有两个：

- 外部引脚 NMI pin，外部设备可以通过这个引脚触发 NMI ，有些服务器甚至提供了 NMI 触发按钮
- 处理器系统总线或者 APIC 串行总线产生的 NMI 消息（包括芯片错误、内存校验错误、总线数据损坏等）

x86 在 IO 端口寄存器 0x70 的 bit7 提供了 NMI_Enable 位，可以如下代码使能、或者禁用 NMI：

```
void NMI_enable(void)
{
    outb(0x70, inb(0x70)&0x7F);
}
void NMI_disable(void)
{
    outb(0x70, inb(0x70)|0x80);
}
```

Linux 内核提供了名为 NMI watchdog 的机制，用于检测系统是否失去响应（也称为 lockup，包括 soft lockup 和 hard lockup），原理是周期性的产生 NMI ，由 NMI handler 响应中断并刷新 hrtimer 定时器，如果一段时间内没有刷新，就表示系统失去了相应，于是调用 panic，超时时间在内核配置里设置，默认是 5 秒。相关代码在内核的 `kernel/watchdog.c` 文件中。

NMI watchdog 依赖 APIC ，所有要将 APIC 编译进内核，启动参数中也不要关闭 APIC 。传统的 x86 架构采用 8259A 芯片处理中断，现在的 x86 架构都引入了 APIC 。可以执行 `cat /proc/interrupts` ，如果输出结果中列出了 IO-APIC-* ，说明系统正在使用 APIC ，如果看到 XT-PIC ，说明系统正在使用 8259A 芯片。

NMI watchdog 的开关是内核启动参数 `nmi_watchdog=[panic,]N` ，也可以在 /etc/sysctl.conf 、/etc/sysctl.d/* 等配置文件中添加内核参数 `kernel.nmi_watchdog=[panic,]N` 。其中 panic 可选，表示 NMI watchdog 超时时产生 panic ，进而可以触发 kdump 。N 可以取值 0~2 ，0 表示禁用 NMI watchdog ，如果要启用 NMI watchdog ，在具有 IO-ACPI 的系统中设为 1 ，在没有 IO-ACPI 的系统中设为 2 。设置成功后，可以看到如下内核信息：

```
$ dmesg | grep NMI
ACPI: LAPIC_NMI (acpi_id[0xff] high edge lint[0x1])
```

```
NMI watchdog: enabled on all CPUs, permanently consumes one hw-PMU counter
```

然后可以看到 NMI 中断计数：

```
$ cat /proc/interrupts  | grep NMI
NMI:        449        207        197        179   Non-maskable interrupts
```

> 因为 NMI 是硬件产生的，所以在虚拟机上测试很可能会失败，内核会报错误信息：NMI watchdog: disable(cpu0): hardware events not enabled

我们可以编写一个模块验证 NMI watchdog 能否正常工作，它的原理是在加载模块时禁用所有中断，这样 NMI handler 就不会响应，也不会刷新定时器，直到超时：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/interrupt.h>

static int __init nmitest_init(void)
{
    printk("nmitest init\n");
    local_irq_disable();
    while(1);
    return 0;
}

static void __exit nmitest_exit(void)
{
    printk("nmitest exit\n");
}

module_init(nmitest_init);
module_exit(nmitest_exit);

MODULE_LICENSE("GPL");
```

系统运行过程中要禁用 NMI watchdog ，可以将 `/proc/sys/kernel/nmi_watchdog` 设为 `0` 。

8. Soft lockup 和 Hard lockup

1913 Words

2018-07-05 00:00 +0000

← Linux 系统如何获取 CPU … systemd 的网络管理 →