

炫技！bug 排查大曝光，涉及Linux 内核的那种

码农的荒岛求生 ITPUB技术小栈 2021-03-01 17:58

那些编程高手除了写代码很厉害之外，调试代码找 bug 更是手到擒来。

编写代码只是程序员的工作之一，调试代码的时间甚至会超过编写代码，之前为大家讲解了很多关于系统、架构、编程等方面的内容，这篇文章就为大家全方位展示一次涉及到内核的 bug 排查过程。

发现问题

话说一天公司服务器报警，登录到机器后发现进程已被“卡死”，常规 GDB 调试没有反应，查找 Log 也没有线索，问题似乎已经无解。

就在这时博主的脑海里浮现出了岛国的。。是的，你猜错了，是岛国的一休哥、柯南弟、国内的包青天、狄仁杰、国外的夏洛克等一众大佬，瞬间有如神助，一定还有办法！是的！

分析问题

先来仔细分析一下，既然进程看上去被卡死，那么如果被卡在用户态，那么该进程 CPU 使用率必然很高（死循环之类）；如果被卡在内核态，这时进程应该正在进行 IO 或者网络通信等，那么 CPU 使用率应该会很低，现在还能查到进程ID，有了进程ID运行 top 命令看一下：

```
# top -cbp 298127
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
298127	root	20	0	109m	1160	844	D	0.0	0.1	0:01.11	server

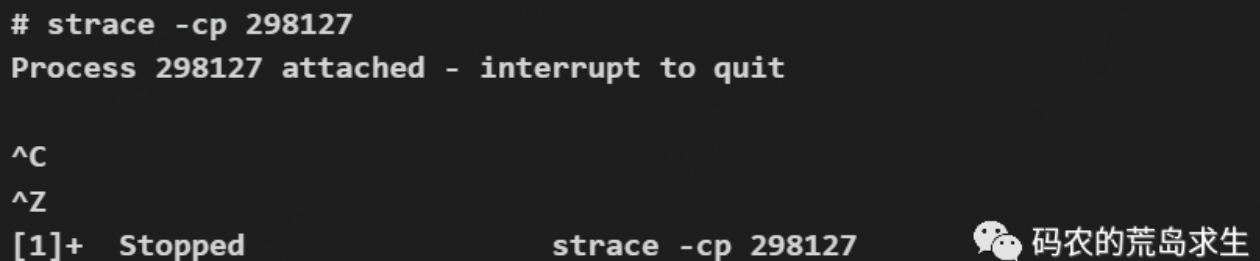
注意 CPU 那一列，显示 CPU 占用率为0%，我们发现此时该进程几乎没有占用 CPU，这基本上是在告诉我们该进程是被卡死在内核态，进程要进入内核态那么就是因为调用了某个阻塞式系统调用导致被操作系统挂起，那么该怎么知道进程调用了什么系统调用呢？

跟踪进程系统调用

strace 命令就用来告诉你这个的，运行 strace 命令来查看一下此时进程调用了什么系统调用：

```
# strace -cp 298127
Process 298127 attached - interrupt to quit

^C
^Z
[1]+  Stopped                  strace -cp 298127
```



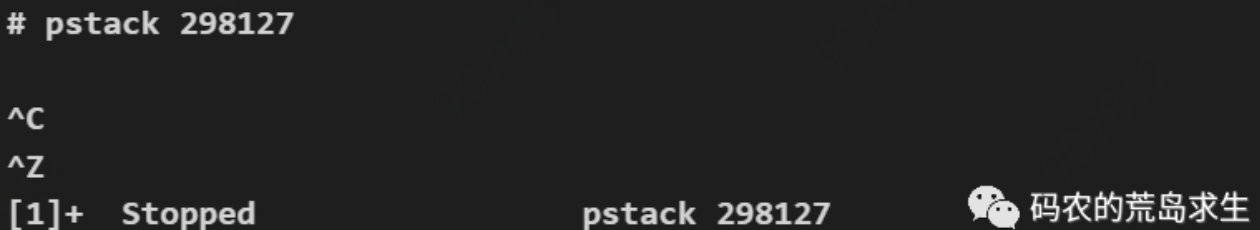
Oops! strace 命令也被卡死了，无奈，再想想还有其它什么办法。。

跟踪进程用户态运行时栈

有了，可以用 pstack 命令，该命令能打印出进程运行时栈信息，虽然该命令不能追踪到内核，但是可以看到用户态最终调用了什么函数，从而推断出调用了什么系统调用，让我们来运行一下：

```
# pstack 298127

^C
^Z
[1]+  Stopped                  pstack 298127
```



和strace一样，pstack 也被卡死了。

现在还能去哪里找线索呢？

古老的ps命令永不过时

我们可以利用 ps 命令来查看进程的运行状态和 WCHAN(waiting channel)。

WCHAN 是什么意思呢？

在 Linux 世界，有问题问男人 (man)，这就是万能的 man 命令，我们使用 man 命令来看一下 ps 展示内容的含义：

```
1 $ man ps
```

运行 man 命令并搜索“WCHAN”，啊哈！最终在“STANDARD FORMAT SPECIFIERS”这一部分中找到了 WCHAN 的含义，是这样写的：

```
wchan      WCHAN      name of the kernel function in which the process is
           sleeping, a "-" if the process is running, or a "*" if
           the process is multi-threaded and ps -T.
           threads.
```

这里清楚的写着 WCHAN 指的是当前进程正阻塞在哪个内核函数上。

OK，我们来运行一下 ps 命令：

```
# ps -flp 298127
F S UID          PID  PPID  C  PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
0 D root          27288 27245  0   80   0 - 28070 rpc_wa 11:57 pts/0      00:00:01 server
```

这里值得注意的是，因为 ps 打印的只是运行ps命令这一时刻相应进程的状态，也就是说运行一次 ps 相当于一次采样，因此你应该多运行几次ps，确保运行结果没有变化，否则只运行一次并且时间足够巧那么有可能会获得到一个错误的线索。


两种进程阻塞状态

从ps打印的结果可以看出，该进程运行状态是D，运行状态D表示什么意思呢？我们再次请教man，发现了这样的信息：

```

D   uninterruptible sleep (usually IO)
R   running or runnable (on run queue)
S   interruptible sleep (waiting for an event to complete)
T   stopped by job control signal
t   stopped by debugger during the tracing
W   paging (not valid since the 2.6.xx kernel)
X   dead (should never be seen)
Z   defunct ("zombie") process, terminated but not reaped by its parent

```

 码农的荒岛求生

原来进程运行状态D表示 `uninterruptible sleep`，不可被打断的 `sleep`，意思是说该进程正在睡觉，就算你拍它一巴掌也不会醒，即该进程当前不响应任何外部信号，此时哪怕 `kill` 命令都杀不掉该进程(除非内核允许该进程接收 `kill` 信号)，直观感受就是该进程被“卡死”了。

与不可被打断的 `sleep` 相对于的是可被打断 `sleep`，从上图看状态为S，此时进程正在阻塞等待某个事件(比如网络数据到来等等)，处于该状态的进程可以接收信号，直观感受就是该进程还有反应。

通过`ps`命令我们可以看到进程状态为D，进一步验证了进程确实被“卡死”了。

那么进程被卡死在了哪里呢？

幸运的是 `WCHAN` 这一列可以告诉你答案。


进程阻塞在哪个内核函数上

上面的`ps`命令 `WCHAN` 这一列显示的是 `rpc_wa`，嗯。。`rpc_wa` 什么呢？看上去是被截断了，不过没关系，我们可以从源头上找到 `wchan` 的完整输出，实际上`ps`等命令也是在这个源头上查找信息并展示出来的，这个源头就是 `proc` 文件系统，`proc` 文件系统记录了内核以及各个进程的运行时信息，我们可以使用最简单的 `cat` 命令，使用 `proc` 后跟进程ID以及`wchan`：

```

# cat /proc/298127/wchan
rpc_wait_bit_killable

```

 码农的荒岛求生

啊哈，我们终于找到进程此时到底卡死在哪里了！

看起来该进程正在等待一个 `RPC` 调用，`RPC` 实际上就是一个进程正在和另一个进程网络通信，尽管我们知道了进程被卡死在了哪里，但是我们依然不知道为什么会卡死在这里。

至此线索似乎中断了。。。

柳暗花明

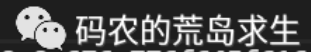
让我们再仔细想一想。

既然进程被卡死了，那么此时进程必然没有位于用户态，不是用户态就肯定是内核态，那么进程怎样才能进入内核态呢？答案很显然是调用了某个系统调用。

那么我们该怎样知道某个进程当前正在调用哪个系统调用呢？

You are lucky dog, Say hi to /proc/***/syscall, 我们同样可以用简单的 cat 命令去 proc 文件系统中查找，使用/proc后跟进程ID+syscall即可。

```
# cat /proc/298127/syscall
262 0xffffffffffff9c 0x20cf6c8 0x7fff97c52710 0x100 0x100 0x676e776f645f616d
```



码农的荒岛求生

WTF。。。这是一串什么鬼东西！

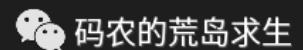
原来这一串看起来不知所云的东西正是系统调用，第一个数字代表系统调用 ID，后面一堆是参数，我们可以不用关心。

从上面的输出我们可以看到调用的是第 262 号系统调用，只有一个数字是没什么意义的，这个数字到底代表那个系统调用呢？

根据内核源码查系统调用

要知道这个数字的含义，我们就需要参考内核代码了，一般在 Linux 系统中必要的内核头文件位于/usr/include目录，在博主 64 位 Linux 机器上，我找到了这个文件：

```
# grep 262 /usr/include/asm/unistd_64.h
#define __NR_newfstatat 262
```



码农的荒岛求生

Gotyou!!! 我们可以看到调用了 newfstatat 系统调用，这个系统调用有什么作用呢？让我们再一次问男人(man命令)：

```
1 $ man newfstatat
```

得到了这样的信息：

```
The underlying system call employed by the glibc fstatat() wrapper function
is actually called fstatat64() or, on some architectures, newfstatat().
```

啊哈！原来是fstatat，这是在读取文件的元信息。


现在我们已经知道了调用什么系统调用，可是一个新的问题再次出现，那就是我们为什么调用这个系统调用后最终会因为等待一个 rpc 被卡死呢？

显然我们需要调用栈信息来验证。

跟踪内核运行时栈

OOOKey，是时候请出重量级工具了，这就是/proc/PID/stack，通过简单的查看这个文件我们就能知道相应进程在内核中的调用栈！！！就问你 Linux 这种设计有没有很厉害，有没有！！！！

```
# cat /proc/298127/stack
[ ] rpc_wait_bit_killable+0x24/0x40 [sunrpc]
[ ] __rpc_execute+0xf5/0x1d0 [sunrpc]
[ ] rpc_execute+0x43/0x50 [sunrpc]
[ ] rpc_run_task+0x75/0x90 [sunrpc]
[ ] rpc_call_sync+0x42/0x70 [sunrpc]
[ ] nfs3_rpc_wrapper.clone.0+0x35/0x80 [nfs]
[ ] nfs3_proc_getattr+0x47/0x90 [nfs]
[ ] __nfs_revalidate_inode+0xcc/0x1f0 [nfs]
[ ] nfs_revalidate_inode+0x36/0x60 [nfs]
[ ] nfs_getattr+0x5f/0x110 [nfs]
[ ] vfs_getattr+0x4e/0x80
[ ] vfs_fstatat+0x70/0x90
[ ] sys_newfstatat+0x24/0x50
[ ] system_call_fastpath+0x16/0x1b
[ ] 0xffffffffffffffff
```


 码农的荒岛求生

这个内核调用栈最终揭开了所有秘密。

真相大白


首先我们来看调用栈的栈顶，栈顶正是 ps 命令 WCHAN 那一列打印出来的，进程在内核中正是因为调用这个函数被卡死的。

```
# cat /proc/298127/stack
[] rpc_wait_bit_killable+0x24/0x40 [sunrpc]
[] __rpc_execute+0xf5/0x1d0 [sunrpc]
[] rpc_execute+0x43/0x50 [sunrpc]
[] rpc_run_task+0x75/0x90 [sunrpc]
[] rpc_call_sync+0x42/0x70 [sunrpc]
[] nfs3_rpc_wrapper.clone.0+0x35/0x80 [nfs]
[] nfs3_proc_getattr+0x47/0x90 [nfs]
[] __nfs_revalidate_inode+0xcc/0x1f0 [nfs]
[] nfs_revalidate_inode+0x36/0x60 [nfs]
[] nfs_getattr+0x5f/0x110 [nfs]
[] vfs_getattr+0x4e/0x80
[] vfs_fstatat+0x70/0x90
[] sys_newfstatat+0x24/0x50
[] system_call_fastpath+0x16/0x1b
[] 0xffffffffffffffff
```

 码农的荒岛求生

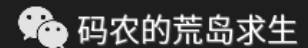
接下来我们从调用栈的最底层看，我们发现了系统调用，印证了正是进程调用这个系统调用而导致卡住的。

```
# cat /proc/298127/stack
[] rpc_wait_bit_killable+0x24/0x40 [sunrpc]
[] __rpc_execute+0xf5/0x1d0 [sunrpc]
[] rpc_execute+0x43/0x50 [sunrpc]
[] rpc_run_task+0x75/0x90 [sunrpc]
[] rpc_call_sync+0x42/0x70 [sunrpc]
[] nfs3_rpc_wrapper.clone.0+0x35/0x80 [nfs]
[] nfs3_proc_getattr+0x47/0x90 [nfs]
[] __nfs_revalidate_inode+0xcc/0x1f0 [nfs]
[] nfs_revalidate_inode+0x36/0x60 [nfs]
[] nfs_getattr+0x5f/0x110 [nfs]
[] vfs_getattr+0x4e/0x80
[] vfs_fstatat+0x70/0x90
[] sys_newfstatat+0x24/0x50
[] system_call_fastpath+0x16/0x1b
[] 0xffffffffffffffff
```

 码农的荒岛求生

那么调用这个系统调用发生了什么呢？我们接着往上看，注意这几行：

```
# cat /proc/298127/stack
[] rpc_wait_bit_killable+0x24/0x40 [sunrpc]
[] __rpc_execute+0xf5/0x1d0 [sunrpc]
[] rpc_execute+0x43/0x50 [sunrpc]
[] rpc_run_task+0x75/0x90 [sunrpc]
[] rpc_call_sync+0x42/0x70 [sunrpc]
[] nfs3_rpc_wrapper.clone.0+0x35/0x80 [nfs]
[] nfs3_proc_getattr+0x47/0x90 [nfs]
[] __nfs_revalidate_inode+0xcc/0x1f0 [nfs]
[] nfs_revalidate_inode+0x36/0x60 [nfs]
[] nfs_getattr+0x5f/0x110 [nfs]
[] vfs_getattr+0x4e/0x80
[] vfs_fstatat+0x70/0x90
[] sys_newfstatat+0x24/0x50
[] system_call_fastpath+0x16/0x1b
[] 0xffffffffffffffff
```



Finally!!! 从调用栈中我们看到了一系列 NFS 相关的函数，NFS全称Network File System，也就是网络文件系统，我们平时挂载(mount)一个远程文件系统就是 NFS来实现的，正是 NFS 进行网络通信才导致在 rpc 上等待，

从内核调用栈我们知道，进程在查询某个远程主机上文件的元数据时因网络问题导致被卡死。

通过这一线索我们最终锁定了出现问题的代码。

总结

本文为大家完整展示了一次 bug 的定位过程，可以看到 Linux 为我们提供了极为丰富的调试工具，当然这离不开 Linux 系统本身优秀的设计思想，那就是将进程和内核的运行时信息通过文件系统提供出来，这极大的方便了问题的排查与定位。

希望本文对大家理解 Linux 系统下问题 debug 有所帮助。

喜欢此内容的人还喜欢

老师电脑故障，小学生开挂5分钟修好，人狠话不多...

了不起的程序员

