strace

跟踪系统调用和信号

补充说明

strace命令 是一个集诊断、调试、统计与一体的工具，我们可以使用strace对应用的系统调用和信号传递的跟踪结果来对应用进行分析，以达到解决问题或者是了解应用工作过程的目的。当然strace与专业的调试工具比如说gdb之类的是没法相比的，因为它不是一个专业的调试器。

strace的最简单的用法就是执行一个指定的命令，在指定的命令结束之后它也就退出了。在命令执行的过程中，strace会记录和解析命令进程的所有系统调用以及这个进程所接收到的所有的信号值。

语法

```
strace  [  -dffhiqrtttTvxx  ] [ -acolumn ] [ -eexpr ] ...
    [ -ofile ] [-ppid ] ...  [ -sstrsize ] [ -uusername ]
    [ -Evar=val ] ...  [ -Evar  ]...
     [command [ arg ...  ] ]

strace  -c  [ -eexpr ] ...  [ -Ooverhead ] [ -Ssortby ]
    [ command [ arg...  ] ]
```

选项

```
-c 统计每一系统调用的所执行的时间,次数和出错的次数等.
-d 输出strace关于标准错误的调试信息.
-f 跟踪由fork调用所产生的子进程.
-ff 如果提供-o filename,则所有进程的跟踪结果输出到相应的filename.pid中,pid是各进程的进程号.
-F 尝试跟踪vfork调用.在-f时,vfork不被跟踪.
-h 输出简要的帮助信息.
-i 输出系统调用的入口指针.
-q 禁止输出关于脱离的消息.
-r 打印出相对时间关于,,每一个系统调用.
-t 在输出中的每一行前加上时间信息.
-tt 在输出中的每一行前加上时间信息,微秒级.
-ttt 微秒级输出,以秒了表示时间.
-T 显示每一调用所耗的时间.
-v 输出所有的系统调用.一些调用关于环境变量,状态,输入输出等调用由于使用频繁,默认不输出.
-V 输出strace的版本信息.
-x 以十六进制形式输出非标准字符串
-xx 所有字符串以十六进制形式输出.
-a column 设置返回值的输出位置.默认 为40.
-e expr 指定一个表达式,用来控制如何跟踪.格式[qualifier=][!]value1[,value2]...
qualifier只能是 trace,abbrev,verbose,raw,signal,read,write其中之一.value是用来限定的符号或数字.默认的 qualifier是 trace.感叹号是否定符号.例如:-eopen等价于 -e trace=open,表示只跟踪open调用.而-etrace!=open 表示跟踪除了open以外的其他调用.有两个特殊的符号 all 和 none. 注意有些shell使用!来执行历史记录里的命令,所以要使用\\.
-e trace=set 只跟踪指定的系统 调用.例如:-e trace=open,close,rean,write表示只跟踪这四个系统调用.默认的为set=all.
```

```
-e trace=file 只跟踪有关文件操作的系统调用.
-e trace=process 只跟踪有关进程控制的系统调用.
-e trace=network 跟踪与网络有关的所有系统调用.
-e strace=signal 跟踪所有与系统信号有关的 系统调用
-e trace=ipc 跟踪所有与进程通讯有关的系统调用
-e abbrev=set 设定strace输出的系统调用的结果集.-v 等与 abbrev=none.默认
为abbrev=all.
-e raw=set 将指定的系统调用的参数以十六进制显示.
-e signal=set 指定跟踪的系统信号.默认为all.如 signal=!SIGIO(或者signal=!io),表示不
跟踪SIGIO信号.
-e read=set 输出从指定文件中读出 的数据.例如: -e read=3,5
-e write=set 输出写入到指定文件中的数据.
-o filename 将strace的输出写入文件filename
-p pid 跟踪指定的进程pid.
-s strsize 指定输出的字符串的最大长度.默认为32.文件名一直全部输出.
-u username 以username的UID和GID执行被跟踪的命令
```

实例

追踪系统调用

现在我们做一个很简单的程序来演示strace的基本用法。这个程序的C语言代码如下：

```
# filename test.c
#include <stdio.h>

int main()
{
    int a;
    scanf("%d", &a);
    printf("%09d\n", a);
    return 0;
}
```

然后我们用gcc -o test test.c编译一下，得到一个可执行的文件test然后用strace调用执行：

```
strace ./test
```

执行期间会要求你输入一个整数，我们输入99，最后得到如下的结果：

```
// 直接执行test的结果
oracle@orainst[orcl]:~ $./test

// 执行的结果
99
000000099

// 通过strace执行test的结果
oracle@orainst[orcl]:~ $strace ./test

// strace的trace结果
```

```
execve("./test", ["./test"], [/* 41 vars */]) = 0
uname({sys="Linux", node="orainst.desktop.mycompany.com", ...}) = 0
brk(0)                                  = 0x8078000
fstat64(3, {st_mode=S_IFREG|0644, st_size=65900, ...}) = 0
old_mmap(NULL, 65900, PROT_READ, MAP_PRIVATE, 3, 0) = 0xbf5ef000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\200X\1"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1571692, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0xbf5ee000
old_mmap(NULL, 1275340, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0xa02000
old_mmap(0xb34000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3,
0x132000) = 0xb34000
old_mmap(0xb37000, 9676, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb37000
close(3)                                = 0
set_thread_area({entry_number:-1 -> 6, base_addr:0xbf5ee740, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1,
seg_not_present:0, useable:1}) = 0
munmap(0xbf5ef000, 65900)               = 0
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xbf5ff000
read(0, 99
"99\n", 1024)                   = 3
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xbf5fe000
write(1, "000000099\n", 10000000099
)             = 10
munmap(0xbf5fe000, 4096)                = 0
exit_group(0)                           = ?
```

从trace结构可以看到，系统首先调用execve开始一个新的进行，接着进行些环境的初始化操作，最后停顿在”read(0,”上面，这也就是执行到了我们的scanf函数，等待我们输入数字呢，在输入完99之后，在调用write函数将格式化后的数值 ” 000000099 ” 输出到屏幕，最后调用exit_group退出进行，完成整个程序的执行过程。

跟踪信号传递

我们还是使用上面的那个test程序，来观察进程接收信号的情况。还是先`strace ./test`，等到等待输入的画面的时候不要输入任何东西，然后打开另外一个窗口，输入如下的命令

```
killall test
```

这时候就能看到我们的程序推出了，最后的trace结果如下：

```
oracle@orainst[orcl]:~
$strace ./test
```

```
execve("./test", ["./test"], [/* 41 vars */]) = 0
uname({sys="Linux", node="orainst.desktop.mycompany.com", ...}) = 0
brk(0)                                  = 0x9ae2000
old_mmap(NULL, 65900, PROT_READ, MAP_PRIVATE, 3, 0) = 0xbf5ef000
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\200X\1"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1571692, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0xbf5ee000
old_mmap(NULL, 1275340, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) = 0x2e9000
old_mmap(0x41b000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 3,
0x132000) = 0x41b000
old_mmap(0x41e000, 9676, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x41e000
close(3)                                = 0
set_thread_area({entry_number:-1 -> 6, base_addr:0xbf5ee740, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1,
seg_not_present:0, useable:1}) = 0
munmap(0xbf5ef000, 65900)               = 0
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xbf5ff000
read(0, 0xbf5ff000, 1024)               = ? ERESTARTSYS (To be restarted)
--- SIGTERM (Terminated) @ 0 (0) ---
+++ killed by SIGTERM +++
```

trace中很清楚的告诉你test进程”+++ killed by SIGTERM +++”

系统调用统计

strace不光能追踪系统调用，通过使用参数-c它还能将进程所有的系统调用做一个统计分析给你，下面就来看看strace的统计，这次我们执行带-c参数的strace

strace -c ./test

最后能得到这样的trace结果：

```
oracle@orainst[orcl]:~
$strace -c ./test
```

```
execve("./test", ["./test"], [/* 41 vars */]) = 0
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 45.90    0.000140           5        27        25 open
 34.43    0.000105           4        24        21 stat64
  7.54    0.000023           5         5           old_mmap
  2.62    0.000008           8         1           munmap
  1.97    0.000006           6         1           uname
  1.97    0.000006           2         3           fstat64
```

```
  1.64    0.000005           3         2         1 read
  1.31    0.000004           2         2           close
  0.98    0.000003           3         1           brk
  0.98    0.000003           3         1           mmap2
  0.66    0.000002           2         1           set_thread_area
------ ----------- ----------- --------- --------- ----------------
100.00    0.000305                    68        47 total
```

这里很清楚的告诉你调用了那些系统函数，调用次数多少，消耗了多少时间等等这些信息，这个对我们分析一个程序来说是非常有用的。

常用参数说明

除了-c参数之外，strace还提供了其他有用的参数给我们，让我们能很方便的得到自己想要的信息，下面就对那些常用的参数一一做个介绍。

重定向输出

参数-o用在将strace的结果输出到文件中，如果不指定-o参数的话，默认的输出设备是STDERR，也就是说使用”-o filename”和 ” 2>filename”的结果是一样的。

```
# 这两个命令都是将strace结果输出到文件test.txt中
strace -c -o test.txt ./test
strace -c ./test  2>test.txt
```

对系统调用进行计时

strace可以使用参数-T将每个系统调用所花费的时间打印出来，每个调用的时间花销现在在调用行最右边的尖括号里面。

```
oracle@orainst[orcl]:~
$strace -T ./test

// 这里只摘录部分结果
read(0, 1
"1\n", 1024)                    = 2 <2.673455>
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
<0.000014>
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xbf5fe000 <0.000017>
write(1, "000000001\n", 10000000001
)             = 10 <0.000016>
munmap(0xbf5fe000, 4096)                 = 0 <0.000020>
exit_group(0)                            = ?
```

系统调用的时间

这是一个很有用的功能，strace会将每次系统调用的发生时间记录下来，只要使用-t/tt/ttt三个参数就可以看到效果了，具体的例子可以自己去尝试。

-t	10:33:04 exit_group(0)	输出结果精确到秒
-tt	10:33:48.159682 exit_group(0)	输出结果精确到微妙
-ttt	1262169244.788478 exit_group(0)	精确到微妙，而且时间表示为unix时间戳

截断输出

-s参数用于指定trace结果的每一行输出的字符串的长度，下面看看test程序中-s参数对结果有什么影响，现指定-s为20，然后在read的是是很我们输入一个超过20个字符的数字串

```
strace -s 20 ./test

read(0, 22222222222222222222222222       // 我们输入的2一共有25个
"22222222222222222222"..., 1024) = 26 // 而我们看到的结果中2只有20个
```

trace一个现有的进程

strace不光能自己初始化一个进程进行trace，还能追踪现有的进程，参数-p就是取这个作用的，用法也很简单，具体如下。

```
strace -p pid
```

综合例子

说了那么多的功能和参数，现在我们来一个实用点的，就是研究下Oracle的lgwr进程，看看这个进程是不是像文档所说的那样没3s钟写一次log文件，考虑到lgwr写日志的触发条件比较多，我们需要找一个空闲的Oracle实例做这个实验。

我们先要得到lgwr进程的pid，运行下面的命令

```
ps -ef|grep lgwr

oracle    5912     1  0 Nov12 ?        00:14:56 ora_lgwr_orcl
```

得到lgwr的pid是5912，现在启动strace，然后将trace的几个输出到lgwr.txt文件中，执行下面的命令

```
strace -tt -s 10 -o lgwr.txt -p 5912
```

过一会之后停止strace，然后查看结果。由于输出的结果比较多，为了方便我们只看Oracle写入log文件时用的pwrite函数的调用

```
grep pwrite\(20 lgwr.txt
```

等等，为什么grep的时候用的是"pwrite(2"呢？，因为我知道我这个机器打开的当前的log文件的句柄编号都是2开始的。具体查找方法是先使用下面的语句找出当前活动的日志文件都有哪些：

```
select member, v$log.status from v$log, v$logfile
where v$log.group#=v$logfile.group#;
```

得到

```
MEMBER                                             STATUS
-------------------------------------------------- ----------------
/db/databases/orcl/redo-01-a/redo-t01-g03-m1.log    INACTIVE
/db/databases/orcl/redo-03-a/redo-t01-g03-m2.log    INACTIVE
/db/databases/orcl/redo-02-a/redo-t01-g02-m1.log    CURRENT
```

```
/db/databases/orcl/redo-04-a/redo-t01-g02-m2.log    CURRENT
/db/databases/orcl/redo-01-a/redo-t01-g01-m1.log    INACTIVE
/db/databases/orcl/redo-03-a/redo-t01-g01-m2.log    INACTIVE
/db/databases/orcl/redo-02-a/redo-t01-g04-m1.log    INACTIVE
/db/databases/orcl/redo-04-a/redo-t01-g04-m2.log    INACTIVE
```

然后到/proc中去找打开文件的句柄：

```
ll /proc/.5912/fd/
```

得到

```
lrwx------    1 oracle   dba            64 Dec 30 10:55 18 ->
/db/databases/orcl/redo-01-a/redo-t01-g01-m1.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 19 ->
/db/databases/orcl/redo-03-a/redo-t01-g01-m2.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 20 ->
/db/databases/orcl/redo-02-a/redo-t01-g02-m1.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 21 ->
/db/databases/orcl/redo-04-a/redo-t01-g02-m2.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 22 ->
/db/databases/orcl/redo-01-a/redo-t01-g03-m1.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 23 ->
/db/databases/orcl/redo-03-a/redo-t01-g03-m2.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 24 ->
/db/databases/orcl/redo-02-a/redo-t01-g04-m1.log
lrwx------    1 oracle   dba            64 Dec 30 10:55 25 ->
/db/databases/orcl/redo-04-a/redo-t01-g04-m2.log
```

现在能看到我机器当前日志文件的句柄分别是20和21。

现在我们得到如下结果

```
11:13:55.603245 pwrite(20, "\1\"\0\0J!"..., 1536, 4363264) = 1536
11:13:55.603569 pwrite(21, "\1\"\0\0J!"..., 1536, 4363264) = 1536
11:13:55.606888 pwrite(20, "\1\"\0\0M!"..., 1536, 4364800) = 1536
11:13:55.607172 pwrite(21, "\1\"\0\0M!"..., 1536, 4364800) = 1536
11:13:55.607934 pwrite(20, "\1\"\0\0P!"..., 1536, 4366336) = 1536
11:13:55.608199 pwrite(21, "\1\"\0\0P!"..., 1536, 4366336) = 1536
11:13:55.610260 pwrite(20, "\1\"\0\0S!"..., 1536, 4367872) = 1536
11:13:55.610530 pwrite(21, "\1\"\0\0S!"..., 1536, 4367872) = 1536
11:14:00.602446 pwrite(20, "\1\"\0\0V!"..., 1536, 4369408) = 1536
11:14:00.602750 pwrite(21, "\1\"\0\0V!"..., 1536, 4369408) = 1536
11:14:00.606386 pwrite(20, "\1\"\0\0Y!"..., 1536, 4370944) = 1536
11:14:00.606676 pwrite(21, "\1\"\0\0Y!"..., 1536, 4370944) = 1536
11:14:00.607900 pwrite(20, "\1\"\0\0\\"..., 1024, 4372480) = 1024
11:14:00.608161 pwrite(21, "\1\"\0\0\\"..., 1024, 4372480) = 1024
11:14:00.608816 pwrite(20, "\1\"\0\0^!"..., 1024, 4373504) = 1024
11:14:00.609071 pwrite(21, "\1\"\0\0^!"..., 1024, 4373504) = 1024
11:14:00.611142 pwrite(20, "\1\"\0\0`!"..., 1536, 4374528) = 1536
11:14:00.611454 pwrite(21, "\1\"\0\0`!"..., 1536, 4374528) = 1536
```

```
11:14:05.602804 pwrite(20, "\1\"\0\0c!"..., 1024, 4376064) = 1024
11:14:05.603119 pwrite(21, "\1\"\0\0c!"..., 1024, 4376064) = 1024
11:14:05.607731 pwrite(20, "\1\"\0\0e!"..., 1024, 4377088) = 1024
11:14:05.608020 pwrite(21, "\1\"\0\0e!"..., 1024, 4377088) = 1024
11:14:05.608690 pwrite(20, "\1\"\0\0g!"..., 1024, 4378112) = 1024
11:14:05.608962 pwrite(21, "\1\"\0\0g!"..., 1024, 4378112) = 1024
11:14:05.611022 pwrite(20, "\1\"\0\0i!"..., 1536, 4379136) = 1536
11:14:05.611283 pwrite(21, "\1\"\0\0i!"..., 1536, 4379136) = 1536
```