# 概述

strace是一个可用于诊断、调试和教学的Linux用户空间跟踪器。我们用它来监控用户空间进程和内核的交互，比如系统调用、信号传递、进程状态变更等。

strace底层使用内核的ptrace特性来实现其功能。

在运维的日常工作中，故障处理和问题诊断是个主要的内容，也是必备的技能。strace作为一种动态跟踪工具，能够帮助运维高效地定位进程和服务故障。它像是一个侦探，通过系统调用的蛛丝马迹，告诉你异常的真相。

参考：

<https://www.cnblogs.com/machangwei-8/p/10388883.html>

# 使用

既然strace是用来跟踪用户空间进程的系统调用和信号的，在进入strace使用的主题之前，我们的先理解什么是系统调用。

关于系统调用：

按维基百科中的解释，在计算机中，系统调用（英语：system call），又称为系统呼叫，指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务。

系统调用提供用户程序与操作系统之间的接口。操作系统的进程空间分为用户空间和内核空间：

操作系统内核直接运行在硬件上，提供设备管理、内存管理、任务调度等功能。

用户空间通过API请求内核空间的服务来完成其功能——内核提供给用户空间的这些API, 就是系统调用。

在Linux系统上，应用代码通过glibc库封装的函数，间接使用系统调用。

Linux内核目前有300多个系统调用，详细的列表可以通过syscalls手册页查看。这些系统调用主要分为几类：

文件和设备访问类 比如open/close/read/write/chmod等

进程管理类 fork/clone/execve/exit/getpid等

信号类 signal/sigaction/kill 等

内存管理 brk/mmap/mlock等

进程间通信IPC shmget/semget \* 信号量，共享内存，消息队列等

网络通信 socket/connect/sendto/sendmsg 等

其他

## 运行模式

我们回到strace的使用上来。strace有两种运行模式。

一种是通过它启动要跟踪的进程。用法很简单，在原本的命令前加上strace即可。比如我们要跟踪 "ls -lh /var/log/messages" 这个命令的执行，可以这样：

strace ls -lh /var/log/messages

另外一种运行模式，是跟踪已经在运行的进程，在不中断进程执行的情况下，理解它在干嘛。这种情况，给strace传递个-p pid 选项即可。

比如，有个在运行的some\_server服务，第一步，查看pid:

pidof some\_server

17553

得到其pid 17553然后就可以用strace跟踪其执行:

strace -p 17553

完成跟踪时，按ctrl + C结束strace即可。

strace有一些选项可以调整其行为，我们这里介绍下其中几个比较常用的，然后通过示例讲解其实际应用效果。

## 常用选项

strace常用选项：

从一个示例命令来看：

strace -tt -T -v -f -e trace=file -o /data/log/strace.log -s 1024 -p 23489

-tt 在每行输出的前面，显示毫秒级别的时间  
 -T 显示每次系统调用所花费的时间  
 -v 对于某些相关调用，把完整的环境变量，文件stat结构等打出来。  
 -f 跟踪目标进程，以及目标进程创建的所有子进程  
 -e 控制要跟踪的事件和跟踪行为,比如指定要跟踪的系统调用名称  
 -o 把strace的输出单独写到指定的文件  
 -s 当系统调用的某个参数是字符串时，最多输出指定长度的内容，默认是32个字节  
 -p 指定要跟踪的进程pid, 要同时跟踪多个pid, 重复多次-p选项即可。

实例：跟踪nginx, 看其启动时都访问了哪些文件

strace -tt -T -f -e trace=file -o /data/log/strace.log -s 1024 ./nginx

部分输出:

输出中，第一列显示的是进程的pid, 接着是毫秒级别的时间，这个是-tt 选项的效果。

每一行的最后一列，显示了该调用所花的时间，是-T选项的结果。

这里的输出只显示和文件访问有关的内容，这是因为我们通过-e trace=file 选项指定了。

strace的使用相当简单，就是执行一个指定的命令比如：

strace -T -tt -s 100 -o /tmp/strace.log CMD

在指定的CMD命令结束之后它也就退出。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。

或者

strace -T -tt -s 100 -o /tmp/strace.log -p PID

对于指定进程号的跟踪，需要Crtl +C 结束调用。

# 参数

-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. 这个参数在排查性能问题时特别有用。

-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.

-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.格式如下:

[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

上面解释strace的参数，其实常用的参数有-T,-tt,-c ,默认-e 为trace=all,有兴趣的朋友可以多试试-e参数来查看不同的参数下日志输出。

# 应用

## 定位进程异常退出

问题：机器上有个叫做run.sh的常驻脚本，运行一分钟后会死掉。需要查出死因。

定位：进程还在运行时，通过ps命令获取其pid, 假设我们得到的pid是24298

strace -o strace.log -tt -p 24298

查看strace.log, 我们在最后2行看到如下内容:

22:47:42.803937 wait4(-1,  <unfinished ...>

22:47:43.228422 +++ killed by SIGKILL +++

这里可以看出，进程是被其他进程用KILL信号杀死的。

实际上，通过分析，我们发现机器上别的服务有个监控脚本，它监控一个也叫做run.sh的进程，当发现run.sh进程数大于2时，就会把它杀死重启。结果导致我们这个run.sh脚本被误杀。

进程被杀退出时，strace会输出killed by SIGX(SIGX代表发送给进程的信号)等，那么，进程自己退出时会输出什么呢？

这里有个叫做test\_exit的程序，其代码如下:

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*\*argv) {

       exit(1);

}

我们strace看下它退出时strace上能看到什么痕迹。

strace -tt -e trace=process -f ./test\_exit

说明: -e trace=process 表示只跟踪和进程管理相关的系统调用。

输出：

23:07:24.672849 execve("./test\_exit", ["./test\_exit"], [/\* 35 vars \*/]) = 0

23:07:24.674665 arch\_prctl(ARCH\_SET\_FS, 0x7f1c0eca7740) = 0

23:07:24.675108 exit\_group(1)           = ?

23:07:24.675259 +++ exited with 1 +++

可以看出，进程自己退出时（调用exit函数，或者从main函数返回）, 最终调用的是exit\_group系统调用， 并且strace会输出exited with X（X为退出码）。

可能有人会疑惑，代码里面明明调用的是exit, 怎么显示为exit\_group?

这是因为这里的exit函数不是系统调用，而是glibc库提供的一个函数，exit函数的调用最终会转化为exit\_group系统调用，它会退出当前进程的所有线程。实际上，有一个叫做\_exit()的系统调用(注意exit前面的下划线), 线程退出时最终会调用它。

## 定位共享内存异常

有个服务启动时报错：

shmget 267264 30097568: Invalid argument

Can not get shm...exit!

错误日志大概告诉我们是获取共享内存出错，通过strace看下：

strace -tt -f -e trace=ipc ./a\_mon\_svr     ../conf/a\_mon\_svr.conf

输出：

22:46:36.351798 shmget(0x5feb, 12000, 0666) = 0

22:46:36.351939 shmat(0, 0, 0)          = ?

Process 21406 attached

22:46:36.355439 shmget(0x41400, 30097568, 0666) = -1 EINVAL (Invalid argument)

shmget 267264 30097568: Invalid argument

Can not get shm...exit!

这里，我们通过-e trace=ipc 选项，让strace只跟踪和进程通信相关的系统调用。

从strace输出，我们知道是shmget系统调用出错了，errno是EINVAL。同样， 查询下shmget手册页，搜索EINVAL的错误码的说明:

EINVAL A new segment was to be created and size < SHMMIN or size > SHMMAX, or no new segment was to be created, a segment with given key existed, but size is greater than the size of that segment

翻译下，shmget设置EINVAL错误码的原因为下列之一：

要创建的共享内存段比 SHMMIN小 (一般是1个字节)

要创建的共享内存段比 SHMMAX 大 (内核参数kernel.shmmax配置)

指定key的共享内存段已存在，其大小和调用shmget时传递的值不同。

从strace输出看，我们要连的共享内存key 0x41400, 指定的大小是30097568字节，明显与第1、2种情况不匹配。那只剩下第三种情况。使用ipcs看下是否真的是大小不匹配：

ipcs  -m | grep 41400

key        shmid      owner      perms      bytes      nattch     status

0x00041400 1015822    root       666        30095516   1

可以看到，已经0x41400这个key已经存在，并且其大小为30095516字节，和我们调用参数中的30097568不匹配，于是产生了这个错误。

在我们这个案例里面，导致共享内存大小不一致的原因，是一组程序中，其中一个编译为32位，另外一个编译为64位,代码里面使用了long这个变长int数据类型。

把两个程序都编译为64解决了这个问题。

## 查看磁盘文件/目录不存在

我们从别的机器copy了个叫做some\_server的软件包过来，可是尝试启动时却报错，根本起不来！

启动命令：

./some\_server ../conf/some\_server.conf

输出:

FATAL: InitLogFile failed iRet: -1!

Init error: -1655

为什么起不来呢？从日志看，似乎是初始化日志文件失败，真相到底怎样呢？我们用strace来看看。

strace -tt -f  ./some\_server ../conf/some\_server.conf

我们注意到，在输出InitLogFile failed错误的前一行，有个open系统调用:

23:14:24.448034 open("/usr/local/apps/some\_server/log//server\_agent.log", O\_RDWR|O\_CREAT|O\_APPEND|O\_LARGEFILE, 0666) = -1 ENOENT (No such file or directory)

它尝试打开文件/usr/local/apps/some\_server/log//server\_agent.log来写(不存在则创建)，可是却出错了，返回码是-1, 系统错误号errorno为ENOENT。 查下open系统调用的手册页：

man 2 open

搜索ENOENT这个错误号errno的解释

ENOENT O\_CREAT  is not set and the named file does not exist.  Or, a directory component in pathname does not exist or is a dangling symbolic link.

这里说得比较清楚，因为我们例子中的open选项指定了O\_CREAT选项，这里errno为ENOENT的原因是日志路径中某个部分不存在或者是一个失效的符号链接。我们来一级一级看下路径中的哪部分不存在：

ls -l /usr/local/apps/some\_server/log

ls: cannot access /usr/local/apps/some\_server/log: No such file or directory

ls -l /usr/local/apps/some\_server

total 8

drwxr-xr-x 2 root users 4096 May 14 23:13 bin

drwxr-xr-x 2 root users 4096 May 14 22:48 conf

原来是log子目录不存在！上层目录都是存在的。手工创建log子目录后，服务就能正常启动了。

回过头来， strace究竟能做什么呢？

它能够打开应用进程的这个黑盒，通过系统调用的线索，告诉你进程大概在干嘛。

# 小结

当应用程序进程出现异常(比如性能间歇性变慢)，我们可以使用strace来跟踪 debug其系统调用，检查程序运行时哪一步耗时比较长，进而找到系统瓶颈。

当然仅仅知道strace工具还是不够的，配合gdb/perf/SystemTap/pstack 等系统工具对应用程序进行“望闻问切”,才能得到我们要的结果。

备注：

1、perf原因kernel支持

2、ftrace  kernel支持可编程

3、systemtap 功能强大，[RedHat](http://www.linuxidc.com/topicnews.aspx?tid=10" \t "https://www.cnblogs.com/machangwei-8/p/_blank)系统支持，对用户态，内核态逻辑都能探查，使用范围更广