1 Strace

1.1. strace 相关基础知识

1.1.1. 什么是 strace

strace 是一个很有用的诊断、学习、调试工具。使用时无需重新编译程序，这也使得可以用来跟踪没有源代码的程序。系统调用和信号是发生在用户空间边界处的事件，检查这些边界事件有助于隔离错误、检查完整性、跟踪程序。

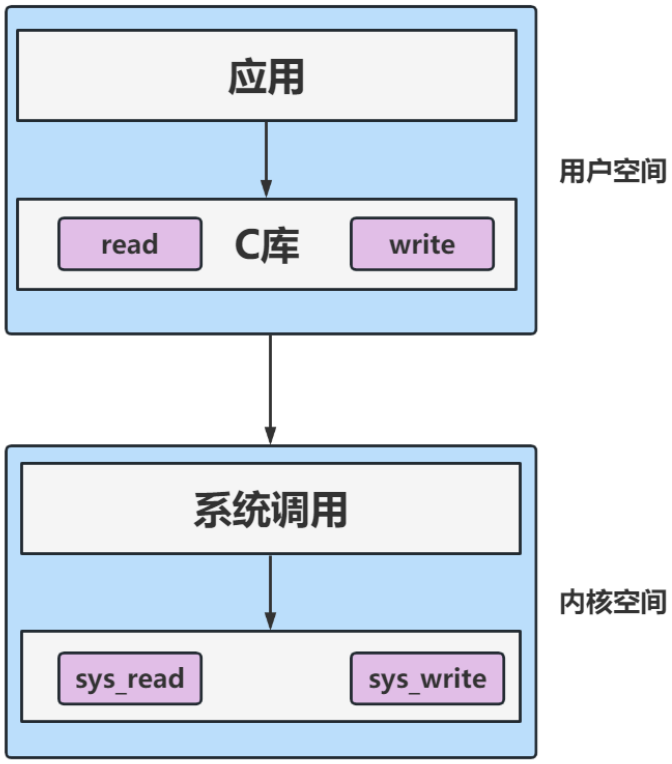
使用 strace 工具来执行程序时，它会纪录程序执行过程中调用的系统调用、接收到的信号。通过查看记录结果，可以知道程序打开了那些文件（open）、打开是否成功、对文件进行了那些读写操作（read、write、ioctl 等）映射了那些内存（mmp）、向系统申请了多少内存等。

1.1.2. 为什么要学会 strace

我们知道，对于应用开发者而言，应用程序的边界是系统调用，进入到系统调用中就是 Linux 内核了。所以，要想拓展分析问题的边界，你首先需要知道该怎么去分析应用程序使用的系统调用函数。对于内核开发者而言，边界同样是系统调用，系统调用之外是应用程序。如果内核开发者想要拓展分析问题的边界，也需要知道如何利用系统调用去追踪应用程序的逻辑。

1.1.3. 什么是系统调用

系统调用是应用程序在执行过程中向操作系统内核申请服务的方法，这可能包含硬件相关的服务、新进程的创建和执行以及进程调度。



1.2. strace 的使用

直接运行 strace -h，查看到它的用法：

```
[root@imx6ull:~]# strace -h
usage: strace [-CdffhiqrtttTvVwxxy] [-I n] [-e expr]...
              [-a column] [-o file] [-s strsize] [-P path]...
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]
   or: strace -c[dfw] [-I n] [-e expr]... [-O overhead] [-S sortby]
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]
```

1.2.1. strace 相关参数

下面是几个常用的选项：

- -f： 除了跟踪当前进程外，还跟踪其子进程。
- -o file：将输出信息写到文件 file 中，而不是显示到标准错误输出（stderr）。
- -p pid：绑定到一个由 pid 对应的正在运行的进程。此参数常用调试后台进程。
- -t：打印各个系统调用被调用是的绝对时间，想观察程序各个部分的执行时间可以使用这个选项。
- -tt：与-t 选项相似，打印的时间精度为μs。
- -r;与选项-t 相似,打印的时间为相对时间。

1.2.2. strace 简单案例

使用 strace 的最简单例子：

```
[root@imx6ull:~]# strace cat /dev/null
execve("/bin/cat", ["cat", "/dev/null"], 0x7ee84dc4 /* 15 vars */) = 0
brk(NULL)                               = 0xcc2000
uname({sysname="Linux", nodename="imx6ull", ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76f2e000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/neon/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l/neon/vfp", 0x7eb866d8) = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/neon/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l/neon", 0x7eb866d8) = -1 ENOENT (No such file or directory)
```

在 strace 的输出结果中，每一行对应一个系统调用；首先是系统调用的名字，紧接着是被包含在括号中的参数，最后是它的返回值，

如第 1 行所示，表示通过系统调用 execve 来建立一个进程，它就是“cat /dev/dull”对应的进程。在控制台中执行各种命令，比如“ls”，“cd”时，都通过系统调用 execve 来建立它们的进程的。最后返回值为 0，表示执行成功。

```
open("/dev/null", O_RDONLY|O_LARGEFILE) = 3
sendfile64(1, 3, NULL, 16777216)        = -1 EINVAL (Invalid argument)
read(3, "", 4096)                       = 0
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

上面例子中，我们忽略中间的打印，看下最后的输出，如上图所示首先打开“/dev/null”文件，这个才是我们“cat /dev/null”命令的真正处理过程，然后读取它的内容。

1.2.3. 使用 strace 来测量程序的执行时间

当我们如果发现某个程序突然执行的很慢，通常需要找出其中哪部分代码执行时间过长，使用 strace 工具可以轻易达到这个目的。

```
strace -r sleep 2
     0.000685 nanosleep({tv_sec=2, tv_nsec=0}, 0x7ec97c10) = 0
     2.001112 exit_group(0)                 = ?
     0.001677 +++ exited with 0 +++
```

如上图实验截图可见，打印了出 sleep 的执行时间。

1.2.4. 使用 strace 来测量程序的执行时间

```
[root@imx6ull:~]# strace -tt -T -e trace=socket,bind,connect ping -c 1 baidu.com
00:02:06.114485 socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3 <0.000313>
00:02:06.116187 connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory) <0.000399>
00:02:06.117757 socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3 <0.000315>
00:02:06.118644 connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory) <0.000359>
00:02:06.127499 socket(AF_INET6, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = 3 <0.000122>
00:02:06.127937 connect(3, {sa_family=AF_INET6, sin6_port=htons(53), inet_pton(AF_INET6, "::1", &sin6_addr), sin6_flowinfo=htonl(0), sin6_scope_id=0}, 28) =
 0 <0.000146>



00:02:11.133076 socket(AF_INET, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = 4 <0.000275>
00:02:11.134088 connect(4, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("127.0.0.1")}, 16) = 0 <0.000276>
```

加上-T 选项后，每行系统调用信息的最右侧会标识出该系统调用的耗时，如<0.000146>。

1.2.5. 统计每个系统调动的耗时、调用次数、失败数

```
[root@imx6ull:~]# strace -c ls
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76fa8068) = 383
wait4(-1, lyx.pacp                                      netease_cloud_music_1.1.0_amd64_binary.tar.xz.1
netease_cloud_music_1.1.0_amd64_binary.tar.xz    netease_cloud_music_1.1.0_amd64_binary.tar.xz.2
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         2           read
  0.00    0.000000           0         2           write
  0.00    0.000000           0        19        16 open
  0.00    0.000000           0         3           close
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         4           ioctl
  0.00    0.000000           0         1           gettimeofday
  0.00    0.000000           0         1           uname
  0.00    0.000000           0         5           mprotect
  0.00    0.000000           0         8           mmap2
  0.00    0.000000           0        16        15 stat64
  0.00    0.000000           0         4           lstat64
  0.00    0.000000           0         4           fstat64
  0.00    0.000000           0         1           getuid32
  0.00    0.000000           0         2           getdents64
  0.00    0.000000           0         1           set_tls
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                    78        32 total
[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], WSTOPPED|WCONTINUED, NULL) = 383
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=383, si_uid=0, si_status=0, si_utime=1, si_stime=3} ---
wait4(-1, 0x7ecf35a8, WNOHANG|WSTOPPED|WCONTINUED, NULL) = -1 ECHILD (No child processes)
```

1.2.6. 跟踪所有网络相关的系统调用

```
[root@imx6ull:~]# strace -e trace=%network ping -c 1 baidu.com
socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)
socket(AF_UNIX, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 3
connect(3, {sa_family=AF_UNIX, sun_path="/var/run/nscd/socket"}, 110) = -1 ENOENT (No such file or directory)
socket(AF_INET6, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET6, sin6_port=htons(53), inet_pton(AF_INET6, "::1", &sin6_addr), sin6_flowinfo=htonl(0), sin6_scope_id=0}, 28) = 0
send(3, "\371\10\1\0\0\1\0\0\0\0\0\0\5baidu\3com\0\0\1\0\1", 27, MSG_NOSIGNAL) = 27
recvfrom(3, "\371\10\201\2\0\0\0\0\0\0\0\0", 1024, 0, {sa_family=AF_INET6, sin6_port=htons(53), inet_pton(AF_INET6, "::1", &sin6_addr), sin6_flowinfo=htonl(
0), sin6_scope_id=0}, [28]) = 12

socket(AF_INET, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = 4
connect(4, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
send(4, "\371\10\1\0\0\1\0\0\0\0\0\0\5baidu\3com\0\0\1\0\1", 27, MSG_NOSIGNAL) = 27
recvfrom(4, "\371\10\201\2\0\0\0\0\0\0\0\0", 1024, 0, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("127.0.0.1")}, [28->16]) = 12
```

-e trace=%network 或者-e trace=%net 是跟踪网络相关系统调用的选项。实际输出的内容远比这个多，这是做了适当省略后的输出。

1.2.7. 跟踪所有信号相关的系统调用

```
[root@imx6ull:~]#  strace -e trace=%signal sh -c echo voidint
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGCHLD, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0
}, 8) = 0
rt_sigaction(SIGCHLD, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=S
A_RESTORER|SA_RESTART, sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
 sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGTSTP, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGTSTP, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
 sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGTTIN, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGTTIN, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
 sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGTTOU, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGTTOU, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
 sa_restorer=0x76de7101}, 8) = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
 sa_restorer=0x76de7101}, 8) = 0
rt_sigaction(SIGCHLD, {sa_handler=0x46834, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x76de7101}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=S
A_RESTORER|SA_RESTART, sa_restorer=0x76de7101}, 8) = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0

rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
+++ exited with 0 +++
```

1.2.8. 跟踪含有文件名参数的系统调用

```
[root@imx6ull:~]# strace -e trace=%file ls /
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76fa8068) = 345
wait4(-1, execve("/bin/ls", ["ls", "/"], 0x7e95adbc /* 15 vars */) = 0
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/neon/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l/neon/vfp", 0x7ee7c6e8) = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/neon/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l/neon", 0x7ee7c6e8) = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l/vfp", 0x7ee7c6e8)  = -1 ENOENT (No such file or directory)
open("/lib/tls/v7l/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/v7l", 0x7ee7c6e8)      = -1 ENOENT (No such file or directory)
open("/lib/tls/neon/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/neon/vfp", 0x7ee7c6e8) = -1 ENOENT (No such file or directory)
open("/lib/tls/neon/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/neon", 0x7ee7c6e8)     = -1 ENOENT (No such file or directory)
open("/lib/tls/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls/vfp", 0x7ee7c6e8)      = -1 ENOENT (No such file or directory)
open("/lib/tls/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/tls", 0x7ee7c6e8)          = -1 ENOENT (No such file or directory)
open("/lib/v7l/neon/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/v7l/neon/vfp", 0x7ee7c6e8) = -1 ENOENT (No such file or directory)
open("/lib/v7l/neon/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/v7l/neon", 0x7ee7c6e8)     = -1 ENOENT (No such file or directory)
open("/lib/v7l/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/v7l/vfp", 0x7ee7c6e8)      = -1 ENOENT (No such file or directory)
open("/lib/v7l/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/v7l", 0x7ee7c6e8)          = -1 ENOENT (No such file or directory)
open("/lib/neon/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/neon/vfp", 0x7ee7c6e8)     = -1 ENOENT (No such file or directory)
open("/lib/neon/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/neon", 0x7ee7c6e8)         = -1 ENOENT (No such file or directory)
open("/lib/vfp/libresolv.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat64("/lib/vfp", 0x7ee7c6e8)          = -1 ENOENT (No such file or directory)
```

```
open("/lib/libresolv.so.2", O_RDONLY|O_CLOEXEC) = 3
open("/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
stat64("/", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
open("/", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_CLOEXEC|O_DIRECTORY) = 3
lstat64("/lost+found", {st_mode=S_IFDIR|0700, st_size=12288, ...}) = 0
lstat64("/lib", {st_mode=S_IFDIR|0755, st_size=2048, ...}) = 0
lstat64("/deepin-boot-maker.exe.2", {st_mode=S_IFREG|0644, st_size=16722432, ...}) = 0
lstat64("/usr", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/tmp", {st_mode=S_IFDIR|S_ISVTX|0777, st_size=360, ...}) = 0
lstat64("/etc", {st_mode=S_IFDIR|0755, st_size=2048, ...}) = 0
lstat64("/libexec", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/bin", {st_mode=S_IFDIR|0755, st_size=2048, ...}) = 0
lstat64("/lib32", {st_mode=S_IFLNK|0777, st_size=3, ...}) = 0
lstat64("/boot", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/samba", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/proc", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat64("/mnt", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/mirrors.php", {st_mode=S_IFREG|0644, st_size=121881, ...}) = 0
lstat64("/sbin", {st_mode=S_IFDIR|0755, st_size=2048, ...}) = 0
lstat64("/deepin-boot-maker.exe", {st_mode=S_IFREG|0644, st_size=16722432, ...}) = 0
lstat64("/dev", {st_mode=S_IFDIR|0755, st_size=3420, ...}) = 0
lstat64("/deepin-boot-maker.exe.3", {st_mode=S_IFREG|0644, st_size=16722432, ...}) = 0
lstat64("/linuxrc", {st_mode=S_IFLNK|0777, st_size=11, ...}) = 0
lstat64("/run", {st_mode=S_IFDIR|0755, st_size=340, ...}) = 0
lstat64("/home", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/var", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/deepin-boot-maker.exe.1", {st_mode=S_IFREG|0644, st_size=16722432, ...}) = 0
lstat64("/sys", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat64("/media", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/opt", {st_mode=S_IFDIR|0755, st_size=1024, ...}) = 0
lstat64("/root", {st_mode=S_IFDIR|0700, st_size=1024, ...}) = 0
bin                     etc                     media                   samba
boot                    home                    mirrors.php             sbin
deepin-boot-maker.exe   lib                     mnt                     sys
deepin-boot-maker.exe.1 lib32                   opt                     tmp
deepin-boot-maker.exe.2 libexec                 proc                    usr
deepin-boot-maker.exe.3 linuxrc                 root                    var
dev                     lost+found              run
+++ exited with 0 +++
```

仔细观察上面的示例可以发现，指定-e trace=%file 选项后，输出的内容拥有一个共同的特点，即均是包含文件名参数的系统调用。

1.2.9. 跟踪已运行进程

```
[root@imx6ull:~]# strace -p $(pidof ntpd)
strace: Process 290 attached
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=75, tv_nsec=994252009}) = 0
gettimeofday({tv_sec=75, tv_usec=994780}, NULL) = 0
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=76, tv_nsec=993985009}) = 0
gettimeofday({tv_sec=76, tv_usec=994503}, NULL) = 0
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=77, tv_nsec=994020009}) = 0
gettimeofday({tv_sec=77, tv_usec=994531}, NULL) = 0
```

此处以进程 ntpd 为例，尝试去跟踪该进程的系统调用。一般线上问题排查场景也是附加到一个正在运行的进程上，观察该进程的系统调用，看是否有错误发生，看是否某个系统调用执行时间过长等。

除了可以指定进程名，我们还可以指定对应进程号。

我们可以先通过 ps 命令查看：

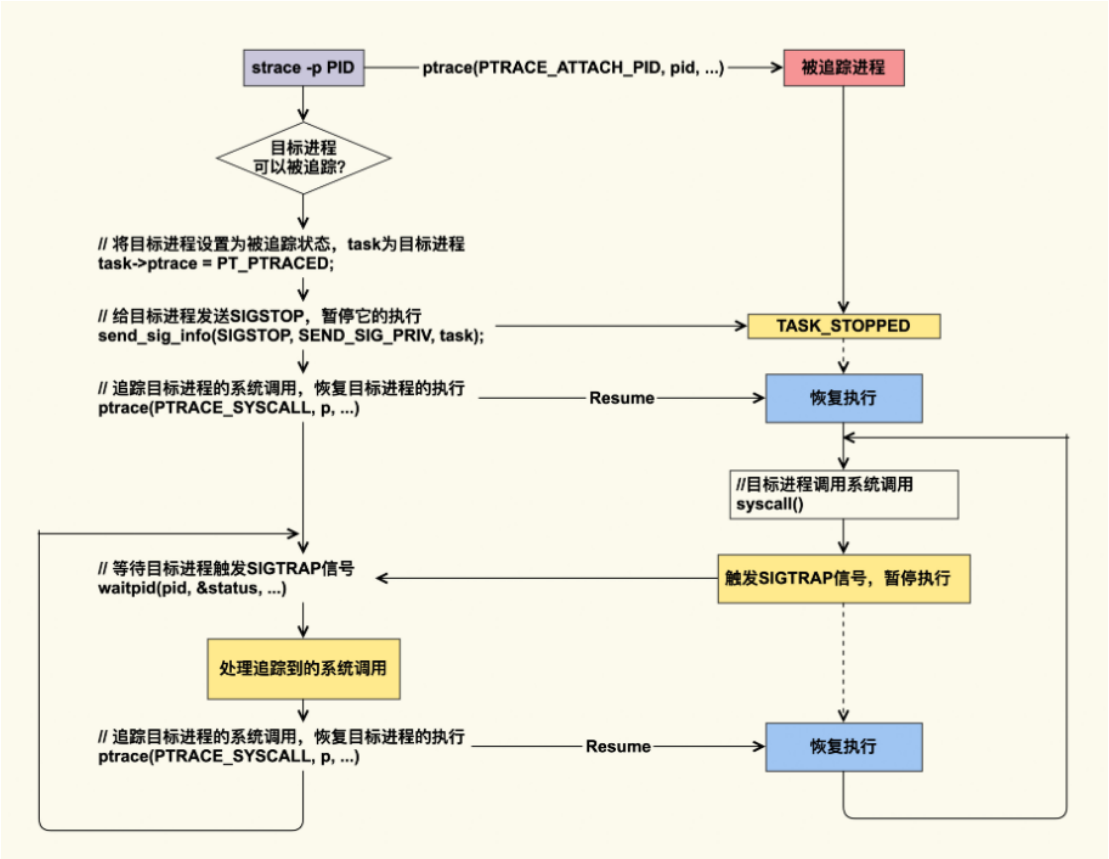
```
290 root     /usr/sbin/ntpd -g
301 root     nginx: master process /usr/sbin/nginx
303 www-data nginx: worker process
306 root     /usr/sbin/sshd
318 root     smbd -D
321 root     {smbd-notifyd} smbd -D
322 root     {cleanupd} smbd -D
323 root     {lpqd} smbd -D
324 root     nmbd -D
329 root     -sh
341 root     ps
```

可以知道，ntpd 对应的进程号为 290。

```
[root@imx6ull:~]# strace -p 290
strace: Process 290 attached
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=250, tv_nsec=994210696}) = 0
gettimeofday({tv_sec=250, tv_usec=994744}, NULL) = 0
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=251, tv_nsec=994032029}) = 0
gettimeofday({tv_sec=251, tv_usec=994545}, NULL) = 0
_newselect(22, [16 17 18 19 20 21], NULL, NULL, NULL) = ? ERESTARTNOHAND (To be restarted if no handler)
--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
sigreturn({mask=[]})                    = -1 EINTR (Interrupted system call)
clock_gettime(CLOCK_REALTIME, {tv_sec=252, tv_nsec=994044363}) = 0
gettimeofday({tv_sec=252, tv_usec=994574}, NULL) = 0
```

1.3. strace 的工作原理

strace 工具的原理如下图所示：



我们从图中可以看到，对于正在运行的进程而言，strace 可以 attach 到目标进程上，这是通过 ptrace 这个系统调用实现的（gdb 工具也是如此）。ptrace 的 PTRACE_SYSCALL 会去追踪目标进程的系统调用；目标进程被追踪后，每次进入 syscall，都会产生 SIGTRAP 信号并暂停执行；追踪者通过目标进程触发的 SIGTRAP 信号，就可以知道目标进程进入了系统调用，然后追踪者会去处理该系统调用，我们用 strace 命令观察到的信息输出就是该处理的结果；追踪者处理完该系统调用后，就会恢复目标进程的执行。被恢复的目标进程会一直执行下去，直到下一个系统调用。

你可以发现，目标进程每执行一次系统调用都会被打断，等 strace 处理完后，目标进程才能继续执行，这就会给目标进程带来比较明显的延迟。因此，在生产环境中我不建议使用该命令，如果你要使用该命令来追踪生产环境的问题，那就一定要做好预案。

假设我们使用 strace 跟踪到，线程延迟抖动是由某一个系统调用耗时长导致的，那么接下来我们该怎么继续追踪呢？这就到了应用开发者和运维人员需要拓展分析边界的时刻了，对内核开发者来说，这才算是分析问题的开始。