# 七种武器问题定位篇之实战Python执行命令僵尸卡死

[mnstory.net](http://mnstory.net/)

结合《七种武器设计篇之设计是自找的》看，前面设计了一个Python调用命令行的封装，我一般在自测上做很多功夫，所以，幸与不幸，还是测试出了问题。

## 构造必现环境

在启动mysql的时候，进程会卡主直到TIMEOUT，子进程是僵尸进程defunct，如下：

root 28058 0.4 0.0 120068 15648 pts/4 S+ 11:41 0:00 \\_ python test.py

root 28455 0.0 0.0 0 0 pts/4 **Z+** 11:41 0:00 \\_ [sh] <defunct>

root 28459 0.0 0.0 22416 1328 pts/4 S+ 11:41 0:00 /bin/sh /var/lib/mysql/bin/mysqld\_safe --datadir=/var/lib/mysql/data --pid-file=/var/lib/mysql/data/host-a0369f033dcb.pid

mysql 28678 1.4 0.5 727476 175156 pts/4 Sl+ 11:41 0:00 \\_ /var/lib/mysql/bin/mysqld --basedir=/var/lib/mysql --datadir=/var/lib/mysql/data --plugin-dir=/var/lib/mysql/lib/plugin --user=mysql --log-error=/var/log/mysql/mysql-error.log --pid-file=/var/lib/mysql

子进程是僵尸进程，那肯定是父进程没有去waitpid，这个问题不是必然的，如果调用的是其他命令，不会出现，所以，我先把命令精简一下，构造一个必现的环境。

test.py代码：

p=subprocess.Popen(**"./b.sh"**, shell=**True**, stdout=subprocess.PIPE, stderr=subprocess.STDOUT, close\_fds=**False**)  
(out, err) = p.communicate(**None**)  
**print** out, err

b.sh代码：

#!/bin/sh

pkill mysqld\_safe

pkill mysqld

/var/lib/mysql/bin/mysqld\_safe --datadir=/var/lib/mysql/data --pid-file=/var/lib/mysql/data/host-a0369f033dcb.pid &

echo "start ok"

## strace工具定位

先strace看下test.py在做啥。

pipe([3, 4]) = 0

fcntl(3, F\_GETFD) = 0

fcntl(3, F\_SETFD, FD\_CLOEXEC) = 0

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

fcntl(3, F\_GETFL) = 0 (flags O\_RDONLY)

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb1b3282000

lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

munmap(0x7fb1b3282000, 4096) = 0

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

read(3, "start ", 6) = 6

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

read(3, "ok\n", 6) = 3

--- SIGCHLD (Child exited) @ 0 (0) ---

read(3, "201", 3) = 3

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

read(3, "31T03:47:18.655324Z mysqld\_sa", 29) = 29

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

read(3, "fe Starting mysqld daemon with d"..., 33) = 33

fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0

lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek)

read(3, "tabases from /var/lib/mysql/data"..., 37) = 33

read(3,

卡在了read函数，从strace跟踪看fd=3是读管道，子进程已经退出了，父进程还在读管道。

## proc文件系统定位

从上面的进程列表可以看出， 28459进程是28058的孙子进程，既然28058卡在读管道上，那孙子进程是否会有相应的写管道未CLOSE？我们查看一下：

# lh /proc/28058/fd

total 0

lrwx------ 1 root root 64 Mar 31 14:31 0 -> /dev/pts/4

lrwx------ 1 root root 64 Mar 31 14:31 1 -> /dev/pts/4

lr-x------ 1 root root 64 Mar 31 14:33 11 -> /dev/urandom

lrwx------ 1 root root 64 Mar 31 14:31 2 -> /dev/pts/4

lr-x------ 1 root root 64 Mar 31 14:31 3 -> pipe:[44876238]

# lh /proc/28459/fd

total 0

lr-x------ 1 root root 64 Mar 31 14:33 0 -> /dev/null

l-wx------ 1 root root 64 Mar 31 14:33 1 -> /dev/null

lr-x------ 1 root root 64 Mar 31 14:33 10 -> /var/lib/mysql/bin/mysqld\_safe\*

lr-x------ 1 root root 64 Mar 31 14:33 11 -> /dev/null

l-wx------ 1 root root 64 Mar 31 14:33 12 -> pipe:[44876238]

l-wx------ 1 root root 64 Mar 31 14:33 13 -> pipe:[44876238]

l-wx------ 1 root root 64 Mar 31 14:31 2 -> /dev/null

## GDB工具验证

的确，孙子继承了咱们的句柄，我们尝试关闭孙子进程继承的管道看看：

# gdb -p 28459

(gdb) call close(12)

$1 = 0

(gdb) call close(13)

$2 = 0

28058进程的终于往下走了，从fork到我们close管道另一端，耗时690s：

#strace -Ttt python test.py

14:31:25.203008 fstat(3, {st\_mode=S\_IFIFO|0600, st\_size=0, ...}) = 0 <0.000004>

14:31:25.203040 lseek(3, 0, SEEK\_CUR) = -1 ESPIPE (Illegal seek) <0.000003>

14:31:25.203064 read(3, "tabases from /var/lib/mysql/data"..., 37) = 33 <0.000005>

14:31:25.203092 read(3,

"", 4) = 0 <**690**.316968>

这里，等待了很久没有继续，现在终于结束了

14:42:55.520112 close(3) = 0 <0.000010>

14:42:55.520203 wait4(28455, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 28455 <0.000020>

14:42:55.520275 write(1, "start ok\n2017-03-31T06:31:25.182"..., 282start ok

问题确认，造成python 执行命令卡死的原因是管道读写句柄继承了，然而继承端并没有关闭管道。

## 原因分析

我们知道，linux句柄会继承是个好事也是个头疼的问题，很少有人记得加：FD\_CLOEXEC或SOCK\_CLOEXEC，于是，我在做supervisor模块的时候，特别处理过类似问题，处理办法很暴力，直接在fork后exec前关闭句柄：

**int** closeAllfds(**int** bIngoreDftFD) {  
 **struct** rlimit rl;  
 **int** closeCnt = 0;  
  
 **if**(-1 == getrlimit(RLIMIT\_NOFILE, &rl)) {  
 lerror(**"getrlimit RLIMIT\_NOFILE failed %d:%s\n"**, errno, strerror(errno));  
 **return** -1;  
 }  
 **if**(rl.rlim\_max == RLIM\_INFINITY) {  
 *//If many files were opened and then this limit was reduced to 1024,   
 //we may not close all file descriptors.* rl.rlim\_max = 1024;  
 }  
  
 **int** fd = 0;  
 **while**(fd < (**int**)rl.rlim\_max) {  
 **if**(!bIngoreDftFD || (fd != STDIN\_FILENO && fd != STDOUT\_FILENO && fd != STDERR\_FILENO)) {  
 **if**(-1 == close(fd)) {  
 **if**(EINTR == errno) {  
 **continue**; *//try again* }  
 **if**(EBADF != errno) {  
 lerror(**"close fd %d failed %d:%s\n"**, fd, errno, strerror(errno));  
 }  
 } **else** {  
 ++closeCnt;  
 lerror(**"close fd %d, total count %d\n"**, fd, closeCnt);  
 }  
 }  
 ++fd;  
 }  
  
 **return** closeCnt;  
}

既然subprocess.Popen对象参数里面可以设置close\_fds标记，那为何不生效？

看看subprocess的源码：

**try**:  
 MAXFD = os.sysconf(**"SC\_OPEN\_MAX"**)  
**except**:  
 MAXFD = 256  
  
errpipe\_read, errpipe\_write = self.pipe\_cloexec()  
  
*# Close all other fds, if asked for - after  
# preexec\_fn(), which may open FDs.***if** close\_fds:  
 self.\_close\_fds(but=errpipe\_write)  
  
**def** \_close\_fds(self, but):  
 **if** hasattr(os, **'closerange'**):  
 os.closerange(3, but)  
 os.closerange(but + 1, MAXFD)  
 **else**:  
 **for** i **in** xrange(3, MAXFD):  
 **if** i == but:  
 **continue  
 try**:  
 os.close(i)  
 **except**:  
 **pass**

关闭方法和我的一样暴力，但是有一个but参数，会将写入端的管道排除，子进程其实是没有继承其他句柄的，但是，偏偏就在排除的句柄上，出了问题，真是防不胜防。

再一睹Python库communicate代码，看是否和分析吻合：

**def** \_readerthread(self, fh, buffer):  
 buffer.append(fh.read())  
  
**def** \_communicate(self, input):  
 stdout = **None** *# Return* stderr = **None** *# Return  
  
 # 指定了stdout为PIPE的时候，会开一个线程来读取* **if** self.stdout:  
 stdout = []  
 stdout\_thread = threading.Thread(target=self.\_readerthread,  
 args=(self.stdout, stdout))  
 stdout\_thread.setDaemon(**True**)  
 stdout\_thread.start()  
 **if** self.stderr:  
 stderr = []  
 stderr\_thread = threading.Thread(target=self.\_readerthread,  
 args=(self.stderr, stderr))  
 stderr\_thread.setDaemon(**True**)  
 stderr\_thread.start()  
  
 **if** self.stdin:  
 **if** input **is not None**:  
 **try**:  
 self.stdin.write(input)  
 **except** IOError **as** e:  
 **if** e.errno == errno.EPIPE:  
 *# communicate() should ignore broken pipe error* **pass  
 elif** (e.errno == errno.EINVAL  
 **and** self.poll() **is not None**):  
 *# Issue #19612: stdin.write() fails with EINVAL  
 # if the process already exited before the write* **pass  
 else**:  
 **raise** self.stdin.close()  
  
 *# 主线程会JOIN* **if** self.stdout:  
 stdout\_thread.join()  
 **if** self.stderr:  
 stderr\_thread.join()  
  
 *# All data exchanged. Translate lists into strings.* **if** stdout **is not None**:  
 stdout = stdout[0]  
 **if** stderr **is not None**:  
 stderr = stderr[0]  
  
 *# Translate newlines, if requested. We cannot let the file  
 # object do the translation: It is based on stdio, which is  
 # impossible to combine with select (unless forcing no  
 # buffering).* **if** self.universal\_newlines **and** hasattr(file, **'newlines'**):  
 **if** stdout:  
 stdout = self.\_translate\_newlines(stdout)  
 **if** stderr:  
 stderr = self.\_translate\_newlines(stderr)  
  
 *# 等PIPE读完了，才会waitpid* self.wait()  
 **return** (stdout, stderr)

虽然子进程已经退出了，但是test.py并没有调用wait，因为它被read PIPE卡主了，所以才会出现子进程defunct，而父进程一直不去回收，和现象完全吻合。

## 解决

解决方法比较简单，问题出在脚本执行的时候，不能用后台运行符号&简单了事，需要用daemon命令替代，daemon命令的源码我之前参考过，里面特别干过close fd的事情，所以，将：

/var/lib/mysql/bin/mysqld\_safe --datadir=/var/lib/mysql/data --pid-file=/var/lib/mysql/data/host-a0369f033dcb.pid &

改为：

daemon -U -- /var/lib/mysql/bin/mysqld\_safe --datadir=/var/lib/mysql/data --pid-file=/var/lib/mysql/data/host-a0369f033dcb.pid

即可。

或者，Python里面不要用PIPE方式取STDOUT亦可。

通过调试过程记录，可以看出，也就是一些知识和工具的运用，技巧不多，还在于积累。

2017/4/1