High一下!

酷 壳 – CoolShell

享受编程和技术所带来的快乐 – Coding Your Ambition
(https://coolshell.cn/)

Search …

VFORK 挂掉的一个问题

2014年11月21日 (Https://coolshell.cn/articles/12103.html) 陈皓 (Https://coolshell.cn/articles/author/haoel) 44,972 人阅读

在知乎上，有个人问了这样的一个问题 (https://www.zhihu.com/question/26591968)——为什么vfork的子进程里用return，整个程序会挂掉，而且exit()不会？并给出了如下的代码，下面的代码一运行就挂掉了，但如果把子进程的return改成exit(0)就没事。

我受邀后本来不想回答这个问题的，因为这个问题明显就是RTFM的事，后来，发现这个问题放在那里好长时间，而挂在下面的几个答案又跑偏得比较严重，我觉得可能有些朋友看到那样的答案会被误导，所以就上去回答了一下这个问题。

下面我把问题和我的回答发布在这里，也供更多的人查看。

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void) {
    int var;
    var = 88;
    if ((pid = vfork()) < 0) {
        printf("vfork error");
        exit(-1);
    } else if (pid == 0) { /* 子进程 */
        var++;
        return 0;
    }
    printf("pid=%d, glob=%d, var=%d\n", getpid(), glob, var);
    return 0;
}
```

基础知识

首先说一下fork和vfork的差别：

- fork 是 创建一个子进程，并把父进程的内存数据copy到子进程中。
- vfork是 创建一个子进程，并和父进程的内存数据share一起用。

这两个的差别是，一个是copy，一个是share。（关于fork，可以参看酷壳之前的《一道fork的面试题(https://coolshell.cn/articles/7965.html)》）

你 man vfork 一下，你可以看到，vfork是这样的工作的，

1）保证子进程先执行。
2）当子进程调用exit()或exec()后，父进程往下执行。

那么，为什么要干出一个vfork这个玩意？ 原因在man page也讲得很清楚了：

> *Historic Description*
>
> *Under Linux, fork(2) is implemented using copy-on-write pages, so the only penalty incurred by fork(2) is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child. However, in the bad old days a fork(2) would require making a complete copy of the caller's data space, often needlessly, since usually immediately afterwards an exec(3) is done. Thus, for greater efficiency, BSD introduced the vfork() system call, which did not fully copy the address space of the parent process, but borrowed the parent's memory and thread of control until a call to execve(2) or an exit occurred. The parent process was suspended while the child was using its resources. The use of vfork() was tricky: for example, not modifying data in the parent process depended on knowing which variables are held in a register.*

意思是这样的—— **起初只有**fork，**但是很多程序在**fork**一个子进程后就**exec**一个外部程序，于是**fork**需要**copy**父进程的数据这个动作就变得毫无意了，而且这样干还很重**（注：后来，fork做了优化，详见本文后面），**所以，**BSD **搞出了个父子进程共享的** vfork，**这样成本比较低。因此，**vfork**本就是为了**exec**而生。**

为什么return会挂掉，exit()不会？

从上面我们知道，**结束子进程的调用是**exit()**而不是**return**，如果你在**vfork**中**return**了，那么，这就意味**main()**函数**return**了，注意因为函数栈父子进程共享，所以整个程序的栈就跪了。**

如果你在子进程中return，那么基本是下面的过程：

1）**子进程的**main() **函数** return**了，于是程序的函数栈发生了变化。**

2）**而**main()**函数**return**后，通常会调用** exit()**或相似的函数**（如：_exit()，exitgroup()）

3）**这时，父进程收到子进程**exit()**，开始从**vfork**返回，但是尼玛，老子的栈都被你子进程给**return**干废掉了，你让我怎么执行？**（注：栈会返回一个诡异一个栈地址，对于某些内核版本的实现，直接报“栈错误”就给跪了，然而，对于某些内核版本的实现，于是有可能会再次调用main()，于是进入了一个无限循环的结果，直到vfork 调用返回 error）

好了，现在再回到 return 和 exit，return会释放局部变量，并弹栈，回到上级函数执行。exit直接退掉。如果你用 c++ 你就知道，return会调用局部对象的析构函数，exit不会。（注：exit不是系统调用，是glibc对系统调用 _exit() 或_exitgroup()的封装）

可见，**子进程调用**exit() **没有修改函数栈，所以，父进程得以顺利执行**。

但是！注意！如果你调用 exit() **函数，还是会有问题的，正确的方法应该是调用** _exit() **函数，因为** exit() **函数 会** flush **并** close **所有的 标准** I/O **，这样会导致父进程受到影响。（这个情况在**fork**下也会受到影响，会导致一些被** buffer**的数据被**flush**两次，这里可以参看《一个**fork**的面试题** (https://coolshell.cn/articles/7965.html)**》）**

关于fork的优化

很明显，fork太重，而vfork又太危险，所以，就有人开始优化fork这个系统调用。优化的技术用到了著名的**写时拷贝（**COW**）**。

也就是说，**对于**fork**后并不是马上拷贝内存，而是只有你在需要改变的时候，才会从父进程中拷贝到子进程中，这样**fork**后立马执行**exec**的成本就非常小了**。所以，Linux的Man Page中并不鼓励使用vfork() ——

> *“ It is rather unfortunate that Linux revived this specter from the past. The BSD man page states: “This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of vfork() as it will, in that case, be made synonymous to fork(2).””*

于是，从BSD4.4开始，他们让vfork和fork变成一样的了

但在后来，NetBSD 1.3 又把传统的vfork给捡了回来，说是vfork的性能在 Pentium Pro 200MHz 的机器（这机器好古董啊）上有可以提高几秒钟的性能。详情见——“NetBSD Documentation: Why implement traditional vfork() (https://www.netbsd.org/docs/kernel/vfork.html)”

今天的Linux下，fork和vfork还是各是各的，不过，还是建议你不要用vfork，除非你非常关注性能。

（全文完）

关注CoolShell微信公众账号和微信小程序

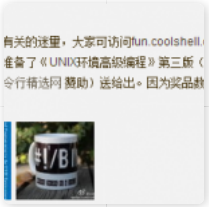
相关文章

(https://coolshell.cn/articles/18360.html)

程序员练级攻略（2018) 与我的专栏 (https://coolshell.cn/articles/18360.html)

(https://coolshell.cn/articles/17998.html)

Linux PID 1 和 Systemd (https://coolshell.cn/articles/17998.html)



(https://coolshell.cn/articles/11847.html)

谜题的答案和活动的心得体会 (https://coolshell.cn/articles/11847.html)

(https://coolshell.cn/articles/9104.html)

sed 简明教程 (https://coolshell.cn/articles/9104.html)

(https://coolshell.cn/articles/9070.html)

AWK 简明教程 (https://coolshell.cn/articles/9070.html)

(https://coolshell.cn/articles/8883.html)

应该知道的Linux技巧 (https://coolshell.cn/articles/8883.html)

(54 人打了分，平均分： 4.37)

C/C++语言 (Https://coolshell.cn/category/proglanguage/cplusplus), Unix/Linux (Https://coolshell.cn/category/operatingsystem/unixlinux), 操作系统 (Https://coolshell.cn/category/operatingsystem)

Fork (Https://coolshell.cn/tag/fork), Linux (Https://coolshell.cn/tag/linux), Unix (Https://coolshell.cn/tag/unix), Vfork (Https://coolshell.cn/tag/vfork)

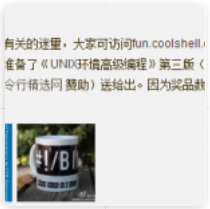
相关文章

(https://coolshell.cn/articles/18360.html)

程序员练级攻略（2018) 与我的专栏 (https://coolshell.cn/articles/18360.html)

(https://coolshell.cn/articles/17998.html)

Linux PID 1 和 Systemd (https://coolshell.cn/articles/17998.html)



(https://coolshell.cn/articles/11847.html)

谜题的答案和活动的心得体会 (https://coolshell.cn/articles/11847.html)

(https://coolshell.cn/articles/9104.html)

sed 简明教程 (https://coolshell.cn/articles/9104.html)

(https://coolshell.cn/articles/9070.html)

AWK 简明教程 (https://coolshell.cn/articles/9070.html)

(https://coolshell.cn/articles/8883.html)

应该知道的Linux技巧 (https://coolshell.cn/articles/8883.html)

《vfork 挂掉的一个问题》的相关评论

int64ago说道：

2014年11月21日 09:17 (https://coolshell.cn/articles/12103.html#comment-1609805)

原来exit不清理资源啊……
那就是等着系统清理咯?

feenn说道：

2014年11月21日 09:24 (https://coolshell.cn/articles/12103.html#comment-1609809)

再次说明不看手册就直接用api的都是危险的行为。
btw：我确实关注性能，所以我大量用vfork。

linpeng1577说道：

2014年11月21日 09:44 (https://coolshell.cn/articles/12103.html#comment-1609820)

@int64ago
释放很多资源，比如cs,ds,文件引用–，会话等等，只是用户栈是和父进程共享的，不会释放，系统清

理，_exit()没有说自己是系统清理

独行猫儿 (http://blog.catscarlet.com)说道：

2014年11月21日 11:15 (https://coolshell.cn/articles/12103.html#comment-1609884)

你这配图好基♂

zhy说道：

2014年11月21日 11:16 (https://coolshell.cn/articles/12103.html#comment-1609886)

又会一招~~

ngcc说道：

2014年11月21日 14:12 (https://coolshell.cn/articles/12103.html#comment-1610014)

APUE中描述：Historically, the exit function has always performed a clean shutdown of the standard I/O library: the fclose function is called for all open streams。其实exit 和 return 作用一样，但是_exit() 和 _Exit() does not perform any flushing of standard I/O buffers

jsxgblcxp说道：

2014年11月21日 14:12 (https://coolshell.cn/articles/12103.html#comment-1610015)

我是来膜拜配图的 哈哈

ngcc说道：

2014年11月21日 14:18 (https://coolshell.cn/articles/12103.html#comment-1610019)

```
1.  #include "apue.h"
2.
3.  int     glob = 6;       /* external variable in initialized da
4.
5.  int
6.  main(void)
7.  {
```

```
 8.      int     var;        /* automatic variable on the stack */
 9.      pid_t   pid;
10.
11.      var = 88;
12.      printf("before vfork\n");   /* we don't flush stdio */
13.      if ((pid = vfork()) &lt; 0) {
14.          err_sys("vfork error");
15.      } else if (pid == 0) {      /* child */
16.          glob++; /* modify parent's variables */
17.          var++;
18.          _exit(0); /* child terminates */
19.                   //此处一般不会调用 exit的，否则影响父进程
20.                   //如果return 就如大Boss说的破坏函数返回地址栈
21.      }
22.      /*
23.       * Parent continues here.
24.       */
25.      printf("pid = %d, glob = %d, var = %d\n", getpid(), glob,
26.      exit(0);
27.  }
```

Williams-Lu说道：

2014年11月21日 15:47 (https://coolshell.cn/articles/12103.html#comment-1610077)

Mark

Mr.Mushroom说道：

2014年11月22日 09:15 (https://coolshell.cn/articles/12103.html#comment-1610766)

耗子老师什么都教啊, RTFM = Read the Fucking Manuel.

wks (https://github.com/wks)说道：

2014年11月22日 22:32 (https://coolshell.cn/articles/12103.html#comment-1611334)

我看的man page是这个版本：http://man7.org/linux/man-pages/man2/vfork.2.html

man page里的这句话让我很不安：“… except that the behavior is undefined if the process created by vfork() either modifies any data other than a variable of type pid_t used to store the return value from vfork() …”这里，“any data”以及“variable”这些词定义得很含糊。variable是C这样的高级语言才有的概念，汇编看到的就只有寄存器和内存。即使用C语言，优化器也可以很自由地决定变量究竟是存放在

寄存器里还是内存里。如果C语言调用exec这样的函数，而参数是表达式，那么编译器还是会生成对参数求值的代码，这其中难免会修改栈上的数据。同时，由于calling convention，编译器会将某些寄存器的值腾出来供传入变量用，同时也会把相应的寄存器的旧值spill到栈上。这其中，“modify any data”几乎任何地方都可能发生。

更令人不安的就是另外一句话：“The use of vfork() was tricky: for example, not modifying data in the parent process depended on knowing which variables were held in a register.”这对编译器的要求太高了。看样子编译器不把vfork变成“内置函数”都没办法。

死神的微笑 (http://cy198706.com)说道：

2014年11月24日 10:24 (https://coolshell.cn/articles/12103.html#comment-1612436)

其实想到fork和vfork的区别就应该明白了。。

cpplog (http://www.cpplog.com)说道：

2014年11月24日 18:01 (https://coolshell.cn/articles/12103.html#comment-1612758)

学习了，耗子老师的每篇文章都让人读后，有所收获。

ChanneW (http://pvq.me)说道：

2014年11月24日 23:34 (https://coolshell.cn/articles/12103.html#comment-1612894)

这配图好有 feel

TinyHui.Wang说道：

2014年11月25日 11:39 (https://coolshell.cn/articles/12103.html#comment-1613444)

你看，英语多重要。man已经写的很详细了。

terry (http://www.gbtags.com)说道：

2014年11月25日 17:51 (https://coolshell.cn/articles/12103.html#comment-1613620)

楼主讲述的很详细，赞

Zenga说道：

2014年11月28日 16:40 (https://coolshell.cn/articles/12103.html#comment-1616104)

现在fork（）应该已经使用了copy-on-wright了吧？所以绝大部分情况下都不需要使用Vfork了

lxyscls说道：

2014年11月28日 17:52 (https://coolshell.cn/articles/12103.html#comment-1616174)

1024个赞

vulcan说道：

2014年11月29日 09:44 (https://coolshell.cn/articles/12103.html#comment-1616650)

关于”child-runs-first”,可参考http://yarchive.net/comp/linux/child-runs-first.html

jxncjl说道：

2014年12月04日 21:49 (https://coolshell.cn/articles/12103.html#comment-1621228)

```
#include
#include
#include
int glob = 6;

int main()
{
int var, v;
var = 5;
v = 88;
pid_t pid;
if((pid = vfork())<0)
{
printf("fork error…");
exit(-1);
```

```
}
else if(pid == 0)
{
v++;
var++;
glob++;
printf("Child procss = %d; v=%d; var=%d; glob=%d\n", getpid(),v,var,glob);
return 0;
}
printf("Parent process = %d;v=%d; pid = %d; var=%d; glob=%d\n", getpid(), v, pid,var,glob);

return 0;
}
```

按照LZ的解释，vfork出的子进程和父进程是共享同一片物理内存区域的，当子进程return后，原来main内存栈就会被摧毁了，之前在该栈中存放的局部变量，如var,v,pid的值都不能保证是原来的值，那为何我每次运行该程序，发现父进行打印出v，pid的值都还是原来的，而v的值却是每次都变动，不科学啊。
运行结果如下所示：

```
seed@seed-desktop:/tmp$ ./a.out
Child procss = 8088; v=89; var=6; glob=7
Parent process = 8087;v=1; pid = 8088; var=-1208462364; glob=7
段错误
seed@seed-desktop:/tmp$ ./a.out
Child procss = 8090; v=89; var=6; glob=7
Parent process = 8089;v=1; pid = 8090; var=-1209146396; glob=7
段错误
```

突然还有一个问题挺困惑的：对于fork，vfork的赋值给pid。对于pid=fork()我是这么理解了：因为fork是子进程拷贝了父进程的一个备份，所以虽然pid在栈中存放的虚拟地址是一样的，但地址映射到不同的物理页，所以在子进程中读取pid的值就是0，而在父进程读取pid的值就为一个常数（即子进程的进程号）。
而pid=vfork()就不好理解了，因为父子进程是共享同一片内存区域的，所以映射到同一个物理页，然而子进程读取pid的值依然为0，这怎么解释呢？然道专门为pid的内存虚拟地址专门开辟了一个物理页给子进程吗？

swpustc (http://swpustc.wicp.net)说道：

2014年12月07日 17:45 (https://coolshell.cn/articles/12103.html#comment-1623597)

@jxncjl
函数的返回值在EAX(RAX)寄存器中，优化过的程序应该直接读的寄存器，除非pid声明为volatile

everett说道：

2014年12月15日 19:57 (https://coolshell.cn/articles/12103.html#comment-1630186)

一年多以前遇到这个问题了，一直没有解决，后来慢慢就淡忘了，今天看到您的博文，让我眼前一亮，立刻翻出一年多前留下的测试代码，终于可以给代码打上“已解决”了。

谢谢您

轩脉刃说道：

2014年12月18日 09:21 (https://coolshell.cn/articles/12103.html#comment-1631463)

学习

周帅印说道：

2015年01月09日 00:01 (https://coolshell.cn/articles/12103.html#comment-1642368)

受教了。。。不过对return和exit的区别没有说太清楚，貌似系统在main()函数return后（弹栈）后会再调用exit().

wanwiy说道：

2015年01月26日 11:59 (https://coolshell.cn/articles/12103.html#comment-1650232)

受教了，程序员应该有的态度。

codefucker说道：

2015年02月11日 12:26 (https://coolshell.cn/articles/12103.html#comment-1659410)

其实，这本质上就是一个return , exit() , _exit()的特性问题跟权利大小的问题，建议多看看那本《Unix环境高级编程》，着重看《Unix编程：进程间通信》 fork（）部分。

netsniffer说道：

2015年03月11日 17:59 (https://coolshell.cn/articles/12103.html#comment-1673411)

Android4.4之前bionic C库及NDK C库的 vfork实现都有类似的问题，导致一用就挂
原理也曾分析了一下：
http://blog.csdn.net/netsniffer/article/details/27834627 (http://blog.csdn.net/netsniffer/article/details/27834627)

itbokeyun (http://itbokeyun.com/)说道：

2015年04月13日 14:33 (https://coolshell.cn/articles/12103.html#comment-1689475)

RTFM

请阅读该死的使用手册

Pingback： Docker基础技术：Linux CGroup | 星达红 (http://www.xdhcn.com/?p=1058)

Pingback： 由vfork()结合exit()想到的若干问题 | 罗宇平的个人博客 (http://beready.me/%e7%94%b1vfork%e7%bb%93%e5%90%88exit%e6%83%b3%e5%88%b0%e7%9a%84%e8%8b%a5%e5%b9%b2%e9%97%ae%e9%a2%98-3/)

shady说道：

2015年05月04日 14:22 (https://coolshell.cn/articles/12103.html#comment-1700892)

讲解得很清晰，又学习了

Dexuan Cui说道：

2015年05月13日 15:37 (https://coolshell.cn/articles/12103.html#comment-1706635)

我看的是Ubuntu 14.04里面的man vfork。

里面说的很清楚，子进程禁止调用 exit():
The child must not return from the current function or call exit(3), but may call _exit(2).

另：exit()会调用atexit()注册的函数，而 _exit()不会。

liaotonglang (http://quant67.com/)说道：

2015年05月20日 13:25 (https://coolshell.cn/articles/12103.html#comment-1710191)

果然很多问题看看文档就解决了。

Pingback： Docker基础技术：Linux CGroup | 酷 壳 - CoolShell.cn (https://coolshell.cn/articles/17049.html)

Pingback： vfork 挂掉的一个问题 | 待定 (http://hunshijidu.sq227.zuji-258.com/?p=324)

WONDERFUL说道：

2015年11月13日 14:37 (https://coolshell.cn/articles/12103.html#comment-1789281)

为啥会修改父进程的栈空间，return只会把栈中保存的函数返回地址，EBP，ESP，EIP等寄存器值修改，但是栈中被回收的资源（函数返回地址，EBP，局部变量等）所占用内存空间的值并不被擦除，而EBP，ESP，EIP这些寄存器是每个进程独享的（进程切换时会恢复为对应的进程的值），怎么会导致栈空间中保存的返回地址等信息被破坏呢？求解！

Bill Lee说道：

2015年11月24日 09:11 (https://coolshell.cn/articles/12103.html#comment-1793874)

@WONDERFUL
从 main() 返回后，libc 会做最后的 termination, 进行一些像调用 atexit(3) 注册的函数之类的操作

Pingback： Docker基础技术：DeviceMapper – 大耳门 (http://www.fewcoo.com/2015/08/26/docker%e5%9f%ba%e7%a1%80%e6%8a%80%e6%9c%af%ef%bc%9adevicemapper/)

Pingback： Docker基础技术：AUFS – 大耳门 (http://www.fewcoo.com/2015/08/24/docker%e5%9f%ba%e7%a1%80%e6%8a%80%e6%9c%af%ef%bc%9aaufs/)

fromdtor (http://nil)说道：

2016年11月03日 00:37 (https://coolshell.cn/articles/12103.html#comment-1895765)

非常清楚

刘伟说道：

2017年02月10日 15:19 (https://coolshell.cn/articles/12103.html#comment-1912970)

关于使用vfork有一个疑问，使用vfork之后，父进程的状态会变成 ‘d’， 也就是不可打断的睡眠，但是这个只是针对内核态么？用户态还是可以正常打断？

Pingback： Linux PID 1 和 Systemd – IT青年文摘 (https://0x1024.com/?p=54)

Pingback： 【酷壳】 记一次Kubernetes/Docker网络排障 | 大白的平凡世界 (http://www.bigbai.net/2018/12/08/%e3%80%90%e9%85%b7%e5%a3%b3%e3%80%91-%e8%ae%b0%e4%b8%80%e6%ac%a1kubernetes-docker%e7%bd%91%e7%bb%9c%e6%8e%92%e9%9a%9c/)

Pingback： Process Problems Caused by vfork – DDCODE (https://ddcode.net/2019/05/07/process-problems-caused-by-vfork/)

Pingback： Docker基础技术：DeviceMapper – 可達書院 (https://sunflower.keda.io/docker%e5%9f%ba%e7%a1%80%e6%8a%80%e6%9c%af%ef%bc%9adevicemapper)