



下载APP



04 | 理解进程（3）：为什么我在容器中的进程被强制杀死了？

2020-11-23 李程远

容器实战高手课

[进入课程 >](#)**讲述：李程远**

时长 13:49 大小 12.66M



你好，我是程远。

今天我们来讲容器中 init 进程的最后一讲，为什么容器中的进程被强制杀死了。理解了这个问题，能够帮助你更好地管理进程，让容器中的进程可以 graceful shutdown。

我先给你说说，为什么进程管理中做到这点很重要。在实际生产环境中，我们有不少应用在退出的时候需要做一些清理工作，比如清理一些远端的链接，或者是清除一些本地的临时数据。



这样的清理工作，可以尽可能避免远端或者本地的错误发生，比如减少丢包等问题的出现。而这些退出清理的工作，通常是在 SIGTERM 这个信号用户注册的 handler 里进行的。

但是，如果我们的进程收到了 SIGKILL，那应用程序就没机会执行这些清理工作了。这意味着，一旦进程不能 graceful shutdown，就会增加应用的出错率。

所以接下来，我们来重现一下，进程在容器退出时都发生了什么。

场景再现

在容器平台上，你想要停止一个容器，无论是在 Kubernetes 中去删除一个 pod，或者用 Docker 停止一个容器，最后都会用到 Containerd 这个服务。


而 Containerd 在停止容器的时候，就会向容器的 init 进程发送一个 SIGTERM 信号。

我们会发现，在 init 进程退出之后，容器内的其他进程也都立刻退出了。不过不同的是，init 进程收到的是 SIGTERM 信号，而其他进程收到的是 SIGKILL 信号。

在理解进程的 [🔗 第一讲](#)中，我们提到过 SIGKILL 信号是不能被捕获的（catch）的，也就是用户不能注册自己的 handler，而 SIGTERM 信号却允许用户注册自己的 handler，这样的话差别就很大了。

那么，我们就一起来看看当容器退出的时候，如何才能让容器中的进程都收到 SIGTERM 信号，而不是 SIGKILL 信号。

延续前面课程中处理问题的思路，我们同样可以运行一个简单的容器，来重现这个问题，用这里的 [🔗 代码](#)执行一下 make image，然后用 Docker 启动这个容器镜像。

 复制代码

```
1 docker run -d --name fwd_sig registry/fwd_sig:v1 /c-init-sig
```

你会发现，在我们用 docker stop 停止这个容器的时候，如果用 strace 工具来监控，就能看到容器里的 init 进程和另外一个进程收到的信号情况。

在下面的例子里，进程号为 15909 的就是容器里的 init 进程，而进程号为 15959 的是容器里另外一个进程。

在命令输出中我们可以看到，**init 进程（15909）收到的是 SIGTERM 信号，而另外一个进程（15959）收到的果然是 SIGKILL 信号。**

[复制代码](#)

```
1 # ps -ef | grep c-init-sig
2 root      15857 14391  0 06:23 pts/0    00:00:00 docker run -it registry/fwd_si
3 root      15909 15879  0 06:23 pts/0    00:00:00 /c-init-sig
4 root      15959 15909  0 06:23 pts/0    00:00:00 /c-init-sig
5 root      16046 14607  0 06:23 pts/3    00:00:00 grep --color=auto c-init-sig
6
7 # strace -p 15909
8 strace: Process 15909 attached
9 restart_syscall(<... resuming interrupted read ...>) = ? ERESTART_RESTARTBLOCK
10 --- SIGTERM {si_signo=SIGTERM, si_code=SI_USER, si_pid=0, si_uid=0} ---
11 write(1, "received SIGTERM\n", 17)      = 17
12 exit_group(0)                                = ?
13 +++ exited with 0 +++
14
15 # strace -p 15959
16 strace: Process 15959 attached
17 restart_syscall(<... resuming interrupted read ...>) = ?
18 +++ killed by SIGKILL +++
```

知识详解：信号的两个系统调用

我们想要理解刚才的例子，就需要搞懂信号背后的两个系统调用，它们分别是 `kill()` 系统调用和 `signal()` 系统调用。

这里呢，我们可以结合前面讲过的信号来理解这两个系统调用。在容器 `init` 进程的第一讲里，我们介绍过信号的基本概念了，**信号就是 Linux 进程收到的一个通知。**

等你学完如何使用这两个系统调用之后，就会更清楚 Linux 信号是怎么一回事，遇到容器里信号相关的问题，你就能更好地理清思路了。


我还会再给你举个使用函数的例子，帮助你进一步理解进程是如何实现 graceful shutdown 的。

进程对信号的处理其实就包括两个问题，**一个是进程如何发送信号，另一个是进程收到信号后如何处理。**

我们在 Linux 中发送信号的系统调用是 `kill()`，之前很多例子里面我们用的命令 `kill`，它内部的实现就是调用了 `kill()` 这个函数。

下面是 Linux Programmer's Manual 里对 `kill()` 函数的定义。


这个函数有两个参数，一个是 `sig`，代表需要发送哪个信号，比如 `sig` 的值是 15 的话，就是指发送 `SIGTERM`；另一个参数是 `pid`，也就是指信号需要发送给哪个进程，比如值是 1 的话，就是指发送给进程号是 1 的进程。

 复制代码

```
1 NAME
2     kill - send signal to a process
3
4 SYNOPSIS
5     #include <sys/types.h>
6     #include <signal.h>
7
8     int kill(pid_t pid, int sig);
```

我们知道了发送信号的系统调用之后，再来看另一个系统调用，也就是 `signal()` 系统调用这个函数，它可以给信号注册 handler。

下面是 `signal()` 在 Linux Programmer's Manual 里的定义，参数 `signum` 也就是信号的编号，例如数值 15，就是信号 `SIGTERM`；参数 `handler` 是一个函数指针参数，用来注册用户的信号 handler。

 复制代码

```
1 NAME
2     signal - ANSI C signal handling
3
4 SYNOPSIS
5     #include <signal.h>
6     typedef void (*sighandler_t)(int);
7     sighandler_t signal(int signum, sighandler_t handler);
```

在容器 `init` 进程的第一讲里，**我们学过进程对每种信号的处理，包括三个选择：调用系统缺省行为、捕获、忽略。**而这里的选择，其实就是程序中如何去调用 `signal()` 这个系统调

用。

第一个选择就是缺省，如果我们在代码中对某个信号，比如 SIGTERM 信号，不做任何 signal() 相关的系统调用，那么在进程运行的时候，如果接收到信号 SIGTERM，进程就会执行内核中 SIGTERM 信号的缺省代码。

对于 SIGTERM 这个信号来说，它的缺省行为就是进程退出 (terminate) 。


内核中对不同的信号有不同的缺省行为，一般会采用退出 (terminate) ，暂停 (stop) ，忽略 (ignore) 这三种行为中的一种。

那第二个选择捕获又是什么意思呢？

捕获指的就是我们在代码中为某个信号，调用 signal() 注册自己的 handler。这样进程在运行的时候，一旦接收到信号，就不会再去执行内核中的缺省代码，而是会执行通过 signal() 注册的 handler。

比如下面这段代码，我们为 SIGTERM 这个信号注册了一个 handler，在 handler 里只是做了一个打印操作。

那么这个程序在运行的时候，如果收到 SIGTERM 信号，它就不会退出了，而是只在屏幕上显示出"received SIGTERM"。


 复制代码

```
1 void sig_handler(int signo)
2 {
3     if (signo == SIGTERM) {
4         printf("received SIGTERM\n");
5     }
6 }
7
8 int main(int argc, char *argv[])
9
10 {
11     ...
12     signal(SIGTERM, sig_handler);
13     ...
14 }
```

我们再来看看第三个选择，如果我们要让进程“忽略”一个信号，我们就要通过 `signal()` 这个系统调用，为这个信号注册一个特殊的 handler，也就是 `SIG_IGN`。

比如下面的这段代码，就是为 `SIGTERM` 这个信号注册 `SIG_IGN`。

这样操作的效果，就是在程序运行的时候，如果收到 `SIGTERM` 信号，程序既不会退出，也不会再在屏幕上输出 log，而是有什么反应也没有，就像完全没有收到这个信号一样。

 复制代码

```
1 int main(int argc, char *argv[])
2 {
3     ...
4     signal(SIGTERM, SIG_IGN);
5     ...
6 }
```

好了，我们通过讲解 `signal()` 这个系统调用，帮助你回顾了信号处理的三个选择：缺省行为、捕获和忽略。

这里我还想要提醒你一点，**`SIGKILL` 和 `SIGSTOP` 信号是两个特权信号，它们不可以被捕获和忽略，这个特点也反映在 `signal()` 调用上。**

我们可以运行下面的 [这段代码](#)，如果我们用 `signal()` 为 `SIGKILL` 注册 handler，那么它就会返回 `SIG_ERR`，不允许我们做捕获操作。

 复制代码

```
1 # cat reg_sigkill.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <errno.h>
6 #include <signal.h>
7
8 typedef void (*sighandler_t)(int);
9
10 void sig_handler(int signo)
11 {
12     if (signo == SIGKILL) {
13         printf("received SIGKILL\n");
14         exit(0);
15     }
16 }
```

```
15         }
16     }
17
18     int main(int argc, char *argv[])
19     {
20         sighandler_t h_ret;
21
22         h_ret = signal(SIGKILL, sig_handler);
23         if (h_ret == SIG_ERR) {
24             perror("SIG_ERR");
25         }
26         return 0;
27     }
28
29 # ./reg_sigkill
30 SIG_ERR: Invalid argument
```

最后，我用下面🔗这段代码来做个小结。

这段代码里，我们用 `signal()` 对 `SIGTERM` 这个信号做了忽略，捕获以及恢复它的缺省行为，并且每一次都用 `kill()` 系统调用向进程自己发送 `SIGTERM` 信号，这样做可以确认进程对 `SIGTERM` 信号的选择。

[📄 复制代码](#)

```
1 #include <stdio.h>
2 #include <signal.h>
3
4 typedef void (*sighandler_t)(int);
5
6 void sig_handler(int signo)
7 {
8     if (signo == SIGTERM) {
9         printf("received SIGTERM\n\n");
10        // Set SIGTERM handler to default
11        signal(SIGTERM, SIG_DFL);
12    }
13 }
14
15 int main(int argc, char *argv[])
16 {
17     //Ignore SIGTERM, and send SIGTERM
18     // to process itself.
19
20     signal(SIGTERM, SIG_IGN);
21     printf("Ignore SIGTERM\n\n");
22     kill(0, SIGTERM);
23 }
```



```
24      //Catch SIGTERM, and send SIGTERM
25      // to process itself.
26      signal(SIGTERM, sig_handler);
27      printf("Catch SIGTERM\n");
28      kill(0, SIGTERM);
29
30
31      //Default SIGTERM. In sig_handler, it sets
32      //SIGTERM handler back to default one.
33      printf("Default SIGTERM\n");
34      kill(0, SIGTERM);
35
36      return 0;
37 }
```

我们一起来总结一下刚才讲的两个系统调用：

先说说 `kill()` 这个系统调用，它其实很简单，输入两个参数：进程号和信号，就把特定的信号发送给指定的进程了。

再说说 `signal()` 这个调用，它决定了进程收到特定的信号如何来处理，`SIG_DFL` 参数把对应信号恢复为缺省 handler，也可以用自定义的函数作为 handler，或者用 `SIG_IGN` 参数让进程忽略信号。

对于 `SIGKILL` 信号，如果调用 `signal()` 函数，为它注册自定义的 handler，系统就会拒绝。

解决问题

我们在学习了 `kill()` 和 `signal()` 这两个信号相关的系统调用之后，再回到这一讲最初的问题上，为什么在停止一个容器的时候，容器 `init` 进程收到的 `SIGTERM` 信号，而容器中其他进程却会收到 `SIGKILL` 信号呢？

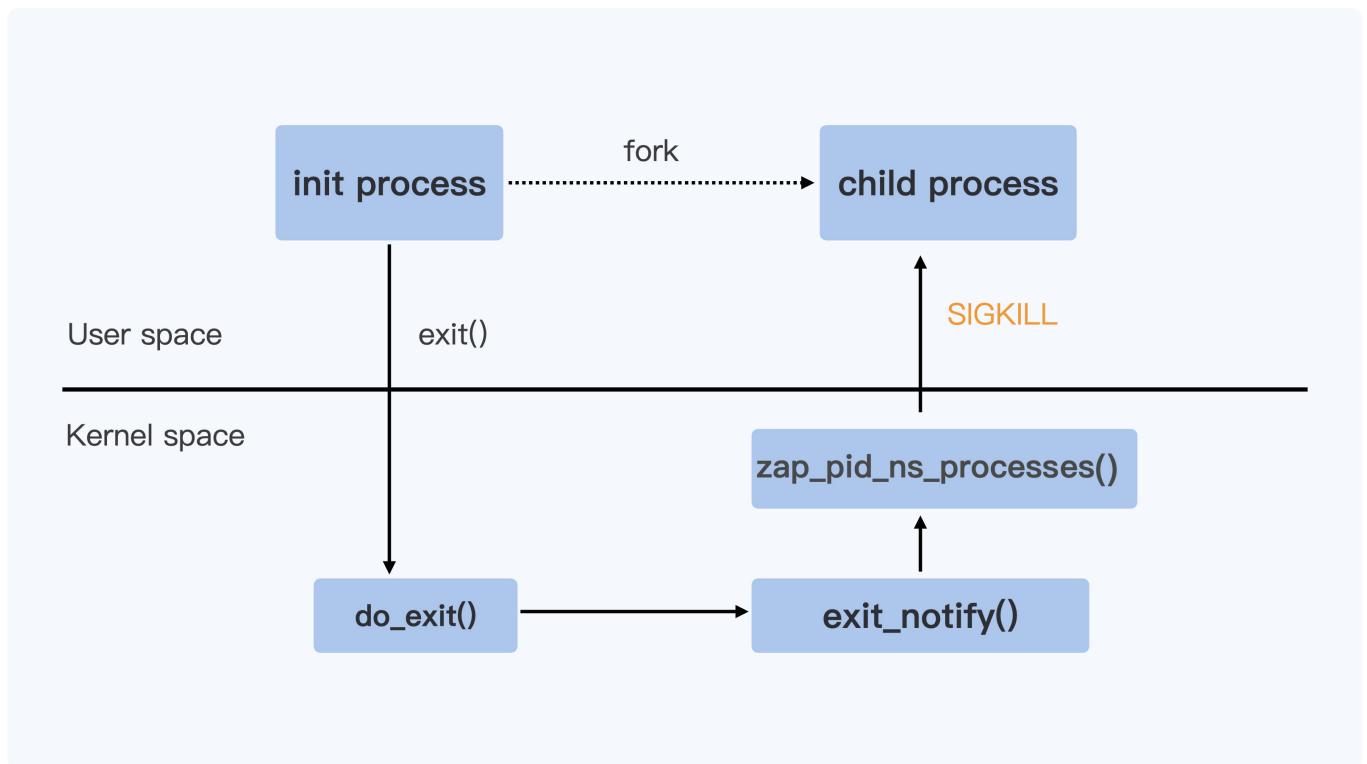
当 Linux 进程收到 `SIGTERM` 信号并且使进程退出，这时 Linux 内核对处理进程退出的入口点就是 `do_exit()` 函数，`do_exit()` 函数中会释放进程的相关资源，比如内存，文件句柄，信号量等等。

Linux 内核对处理进程退出的入口点就是 `do_exit()` 函数，`do_exit()` 函数中会释放进程的相关资源，比如内存，文件句柄，信号量等等。


在做完这些工作之后，它会调用一个 `exit_notify()` 函数，用来通知和这个进程相关的父子进程等。

对于容器来说，还要考虑 Pid Namespace 里的其他进程。这里调用的就是 `zap_pid_ns_processes()` 这个函数，而在这个函数中，如果是处于退出状态的 `init` 进程，它会向 Namespace 中的其他进程都发送一个 `SIGKILL` 信号。

整个流程如下图所示。



你还可以看一下，内核代码是这样的。

 复制代码

```
1      /*
2          * The last thread in the cgroup-init thread group is terminating.
3          * Find remaining pid_ts in the namespace, signal and wait for them
4          * to exit.
5          *
6          * Note: This signals each threads in the namespace - even those that
7          *       belong to the same thread group, To avoid this, we would hav
8          *       to walk the entire tasklist looking a processes in this
9          *       namespace, but that could be unnecessarily expensive if the
10         *       pid namespace has just a few processes. Or we need to
11         *       maintain a tasklist for each pid namespace.
12         *
13         */
```

```
14     rcu_read_lock();
15     read_lock(&tasklist_lock);
16     nr = 2;
17     idr_for_each_entry_continue(&pid_ns->idr, pid, nr) {
18         task = pid_task(pid, PIDTYPE_PID);
19         if (task && !__fatal_signal_pending(task))
20             group_send_sig_info(SIGKILL, SEND_SIG_PRIV, task, PIDT
21     }
22
```

说到这里，我们也就明白为什么容器 init 进程收到的 SIGTERM 信号，而容器中其他进程却会收到 SIGKILL 信号了。

前面我讲过，SIGKILL 是个特权信号（特权信号是 Linux 为 kernel 和超级用户去删除任意进程所保留的，不能被忽略也不能被捕获）。

所以进程收到这个信号后，就立刻退出了，没有机会调用一些释放资源的 handler 之后，再做退出动作。

而 SIGTERM 是可以被捕获的，用户是可以注册自己的 handler 的。因此，容器中的程序在 stop container 的时候，我们更希望进程收到 SIGTERM 信号而不是 SIGKILL 信号。

那在容器被停止的时候，我们该怎么做，才能让容器中的进程收到 SIGTERM 信号呢？

你可能已经想到了，就是让容器 init 进程来转发 SIGTERM 信号。的确是这样，比如 Docker Container 里使用的 tini 作为 init 进程，tini 的代码中就会调用 sigtimedwait() 这个函数来查看自己收到的信号，然后调用 kill() 把信号发给子进程。

我给你举个具体的例子说明，从下面的这段代码中，我们可以看到除了 SIGCHLD 这个信号外，tini 会把其他所有的信号都转发给它的子进程。

[复制代码](#)

```
1  int wait_and_forward_signal(sigset_t const* const parent_sigset_ptr, pid_t co
2
3      siginfo_t sig;
4
5      if (sigtimedwait(parent_sigset_ptr, &sig, &ts) == -1) {
6          switch (errno) {
7      ...
8          }
```

```
9      } else {
10          /* There is a signal to handle here */
11          switch (sig.si_signo) {
12              case SIGCHLD:
13                  /* Special-cased, as we don't forward SIGCHLD.
14                   * fallthrough to reaping processes.
15                   */
16                  PRINT_DEBUG("Received SIGCHLD");
17                  break;
18              default:
19                  PRINT_DEBUG("Passing signal: '%s'", strsignal(
20                      /* Forward anything else */
21                      if (kill(kill_process_group ? -child_pid : chi
22                          if (errno == ESRCH) {
23                              PRINT_WARNING("Child was dead
24                          } else {
25                              PRINT_FATAL("Unexpected error
26
27                              return 1;
28                          }
29                      }
30                      break;
31          }
32      }
33      return 0;
34 }
```

那么我们在这里明确一下，怎么解决停止容器的时候，容器内应用程序被强制杀死的问题呢？

解决的方法就是在容器的 init 进程中对收到的信号做个转发，发送到容器中的其他子进程，这样容器中的所有进程在停止时，都会收到 SIGTERM，而不是 SIGKILL 信号了。

重点小结

这一讲我们要解决的问题是让容器中的进程，在容器停止的时候，有机会 graceful shutdown，而不是收到 SIGKILL 信号而被强制杀死。

首先我们通过对 kill() 和 signal() 这两个系统调用的学习，进一步理解了进程是怎样处理 Linux 信号的，重点是信号在接收处理的三个选择：**忽略，捕获和缺省行为**。

通过代码例子，我们知道 SIGTERM 是可以被忽略和捕获的，但是 SIGKILL 是不可以被忽略和捕获的。

了解这一点以后，我们就找到了问题的解决方向，也就是我们需要在停止容器时，让容器中的应用收到 SIGTERM，而不是 SIGKILL。


具体怎么操作呢？我们可以在容器的 init 进程中对收到的信号做个转发，发送到容器中的其他子进程。这样一来，容器中的所有进程在停止容器时，都会收到 SIGTERM，而不是 SIGKILL 信号了。

我认为，解决 init 进程信号的这类问题其实并不难。

我们只需要先梳理一下和这个问题相关的几个知识点，再写个小程序，让它跑在容器里，稍微做几个试验。然后，我们再看一下内核和 Docker 的源代码，就可以很快得出结论了。

思考题

请你回顾一下基本概念中最后的这段代码，你可以想一想，在不做编译运行的情况下，它的输出是什么？

 复制代码

```
1 #include <stdio.h>
2 #include <signal.h>
3
4 typedef void (*sighandler_t)(int);
5
6 void sig_handler(int signo)
7 {
8     if (signo == SIGTERM) {
9         printf("received SIGTERM\n\n");
10        // Set SIGTERM handler to default
11        signal(SIGTERM, SIG_DFL);
12    }
13 }
14
15 int main(int argc, char *argv[])
16 {
17     //Ignore SIGTERM, and send SIGTERM
18     // to process itself.
19
20     signal(SIGTERM, SIG_IGN);
21     printf("Ignore SIGTERM\n\n");
22     kill(0, SIGTERM);
23
24     //Catch SIGTERM, and send SIGTERM
```

```
25     // to process itself.
26     signal(SIGTERM, sig_handler);
27     printf("Catch SIGTERM\n");
28     kill(0, SIGTERM);
29
30
31     //Default SIGTERM. In sig_handler, it sets
32     //SIGTERM handler back to default one.
33     printf("Default SIGTERM\n");
34     kill(0, SIGTERM);
35
36     return 0;
37 }
```

欢迎留言和我分享你的想法和疑问。如果读完这篇文章有所收获，也欢迎你分享给自己的朋友，共同学习和进步。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | 理解进程（2）：为什么我的容器里有这么多僵尸进程？

下一篇 05 | 容器CPU（1）：怎么限制容器的CPU使用？

精选留言 (19)

写留言



po

2020-11-25

老师，我做了个测试，现象有点迷惑，我打开两个终端，用sleep进行测试，方法和现象如下：

1. 在第一个终端启动sleep，在另外一个终端通过命令去kill，能通过sigterm正常杀掉进程。

```
# strace sleep 30000...
```

展开 ∨

作者回复: @po,

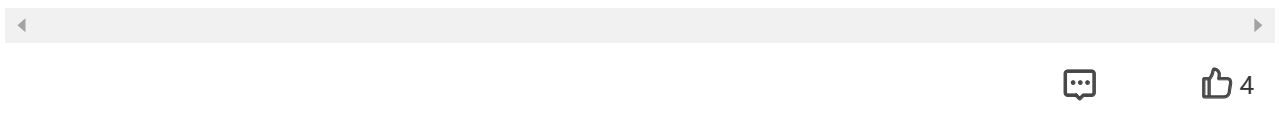
对于第二个问题，我假设sleep进程在宿主机上的pid是2207，你还是可以先查看"cat /proc/2207/status | grep SigCgt"，我的理解是SIGTERM handler应该还是没有注册，那么即使从宿主机上发送SIGTERM给这个容器里的1号进程，那么也是不能杀死的。

"docker stop"在停止容器的时候，先给容器里的1号进程发送SIGTERM, 如果不起作用，那么等待30秒后会发送SIGKILL。我想这个是你看到的现象了。

至于为什么即使在宿主机机上向容器1号进程发送SIGTERM，在1号进程没有注册handler的情况下，不能被杀死的问题（思考题），原因是这样的：

开始要看内核里的那段代码，"`!(force && sig_kernel_only(sig))`"，虽然由不同的namespace发送信号，虽然force是1了，但是sig_kernel_only(sig)对于SIGTERM来说还是0，这里是个&&，那么`!(1 && 0) = 1`。

```
#define sig_kernel_only(sig) signmask(sig, SIG_KERNEL_ONLY_MASK)
#define SIG_KERNEL_ONLY_MASK (\
    rt_sigmask(SIGKILL) | rt_sigmask(SIGSTOP))
```



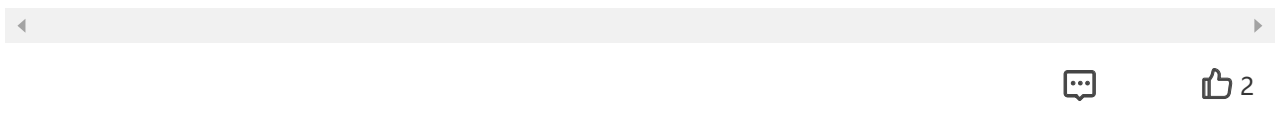
宝仔

2020-11-25

老师，容器的最佳实践一般都是一个容器即一个进程，一般如果按照这种做法，就只需要在应用程序进程中对sigterm信号做捕获并处理就行了吧，无需转发吧

展开 ∨

作者回复: 是的!



上邪忘川

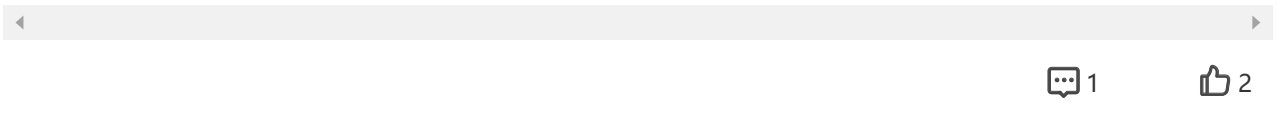
2020-11-23

可以用strace跟踪进程的信号，用法参考<https://www.cnblogs.com/machangwei-8/p/10388883.html>

运维同学表示代码没看懂，哈哈

作者回复: @上邪忘川

strace 主要用来查看程序调用了哪些系统调用已经收到什么信号。



JianXu

2020-11-28

老师，这里的逻辑我还没有理顺。

1. 你说的容器init 进程，是不是就是容器的第一个进程？还有是不是如果我使用docker，容器的第一个进程一定不是我自己的进程，而是tini 进程？

...

展开 ∨

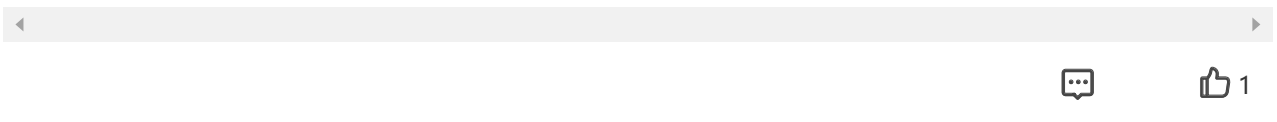
作者回复: > 1

是的, init进程就是容器里的第一个进程。容器里的第一个进程大部分情况应该是我们自己的进程，除非有容器用户有意识的去使用tini进程作为init进程。

> 2

很好的问题。

init 进程自己退出，还是会调用do_exit()的。所以呢，为了保证子进程先收到转发的SIGTERM，类似tini的做法是，自己在收到SIGTERM的时候不退出，转发SIGTERM给子进程，子进程收到SIGTERM退出之后会给父进程发送SIGCHLD，tini是收到SIGCHLD之后主动整个程序退出。



po

2020-11-25

老师，我能提个建议吗？这几天学习容器进程和信号相关的知识点，有点乱，自己理出来也好像怪怪的，你能不能画个图，把进程的信号相关的给我们捋一遍呢？还有我们程序代码该如何更好的设计能给一点建议吗？感觉用tini这种方式改动有点大，之前我们一直都是应用程序作为PID1来运行的，好像也没啥问题。谢谢

展开 ∨

作者回复: @po,

谢谢你这几天提的问题，看的出来，你做了很多的测试，也有很多的思考！我们可以一起来，先把你前面的问题逐个理清了。

你说的画图来捋一遍信号概念的，这个我会考虑的（目前我需要先完成课程后面章节的内容 :-), 你

也可以给我一些更详细的建议，可能可以和你问的具体问题想结合。

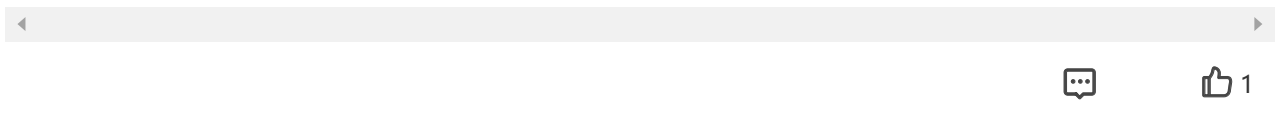
> 之前我们一直都是应用程序作为PID1来运行的，好像也没啥问题

信号对容器中进程的影响的多少，也有多方面的原因，比如程序本身对错误的容忍度比较高，容器建立删除的频率不高，那么也就看不出有什么影响。

如果你的程序的容器化程度较高，几乎是一个容器一个进程的程度，那么不需要考虑用tini来做改动。

我觉得容器里的init进程，应该是具备这些信号处理的能力：

1. 至少转发SIGTERM给容器里其他的关键子进程。
2. 能够接受到外部的SIGTERM信号而退出，（这里可以是通过注册SIGTERM handler, 也可以像tini一样先转发SIGTERM 给子进程，然后收到SIGCHLD后自己主动退出）
3. 具有回收zombie进程的能力。



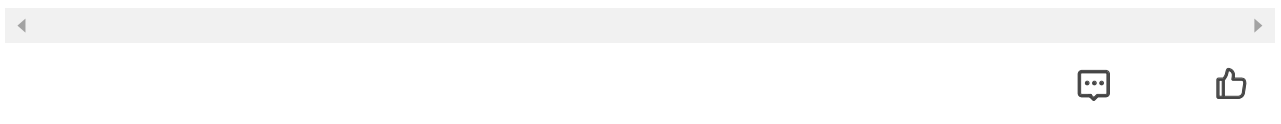
Alery

2020-12-01

老师，我有个疑问哈，tini没有注册SIGTERM，按照前面说的，内核是不会把这个信号发送给tini进程的，为啥它又能接收所有的信号(除了SIGCHLD)并转发给子进程呢？我对这块的理解的不是很清晰，望指教。

作者回复: 很好的问题！

因为在tini里调用的sigtimedwait()系统调用，直接把发送给tini的信号先截了下来，这时候tini有没有SIGTERM的handler就没有关系了。



Alery

2020-12-01

老师，请教一个问题，tini 会把其他所有的信号都转发给它的子进程，假如我的子进程又创建了子进程(也就是tini的孙子进程)，tini会把信号转发给孙子进程吗？

展开 ∨

作者回复: 我们可以从tini转发信号的代码看一下。如果 “kill_process_group” 没有设置，为0时，这也是tini缺省的配置，那么SIGTERM只会转发给子进程，而子子进程就不会收到转发的SIGTERM。当子进程退出的时候，子子进程就会收到SIGKILL。

而如果kill_process_group > 0的时候，同时子进程与子进程在同一个process group的时候（缺省fork出来的子进程会和父进程在同一个process group），那么子进程就会收到SIGTERM

```
if (kill(kill_process_group ? -child_pid : child_pid, sig.si_signo))
```



胖胖虎

2020-11-30

简单总结了下，子进程被kill杀死的原因是，父进程在退出时，执行do_exit中，由于是cgroup_init 组的进程，因此向所有的子进程发送了sigkill信号。而导致这个的原因是，一般情况下，容器起来的第一个进程都不是专业的init进程，没有考虑过这些细节问题。由于正常情况下，父进程被终结，信号不会传递到子进程，exit时也不会给子进程发终结命令。这会导致多进程容器在关闭时，无法被终止。为了保证容器能够被正常终结。设计者在d...
展开

作者回复: @胖胖虎， 很好的总结！



小鱼

2020-11-30

我的容器的init进程是通过bash拉起的Linux守护进程，然后守护进程会创建子进程一个MySQL实例，为了优雅退出，我该如何改写init进程呢？
展开

作者回复: @小鱼，

你这里的“Linux守护进程”指的是mysqld吗？如果是只是想mysqld graceful shutdown, 可以用tini来启动mysqld, 不过还需要看你的bash里有没有做其他的工作。



JianXu

2020-11-28

CY，能帮忙解释一下我们公司生产环境在容器image patching 过程中应用程序受影响的事情吗。

1. 我们的胖容器肯定是多进程的，那当容器收到kill 命令的时候，我们现在也是子容器都被SIGKill 吗？还是我们其实都是配置了Init 进程，而init 进程其实都像文中说的转发了 ...

展开 ∨

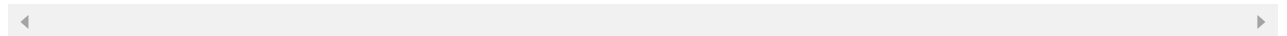
作者回复: @JianXu,

你说的情况是这样的，

胖容器的init进程其实是一个bash脚本run.sh, 由它来启动jvm的程序。

但是run.sh本身没有注册SIGTERM handler, 也不forward SIGTERM给子进程jvm。

当stop容器的时候，run.sh先收到一个SIGTERM, run.sh没有注册SIGTERM, 所以呢对SIGTERM没有反应，containrd过30秒，会发SIGKILL给run.sh, 这样run.sh退出do_exit(), 在退出的时候同样给子进程jvm程序发送了SIGKILL而不是SIGTERM。其实呢，jvm的程序是注册了SIGTERM handler的，但是没有机会调用handler了。



lihome

2020-11-25

(评论不支持多级，故再次整理发出)

老师能否将编译测试源码的过程放置到基于CentOS的容器内部

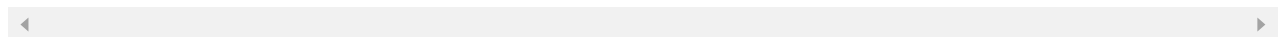
目前使用 make image 构建镜像：

编译过程中产生的的可执行文件是基于宿主机的(然后COPY到容器了)

运行环境是基于Centos的容器...

展开 ∨

作者回复: 嗯，我可以把我自己编译的binary上传了， make image的时候可以直接打包到镜像。



po

2020-11-24

老师，如果通过tini转发信号给子进程，那么子子进程是收到子进程的信号吧？那么子子进程收到的信号是sigkill还是sigterm呢？

展开 ∨

作者回复: @po 非常好的问题。

我们可以从tini转发信号的代码看一下。如果 “kill_process_group” 没有设置，为0时，这也是tini缺省的配置，那么SIGTERM只会转发给子进程，而子子进程就不会收到转发的SIGTERM。

当子进程退出的时候，子子进程就会收到SIGKILL。

而如果kill_process_group > 0的时候，同时子进程与子进程在同一个process group的时候（缺省fork出来的子进程会和父进程在同一个process group），那么子进程就会收到SIGTERM

```
if (kill(kill_process_group ? -child_pid : child_pid, sig.si_signo))
```



朱雯

2020-11-24

看完这篇课程后，兴冲冲的看生产环境的init进程是什么，有没有机会改成tini，结果一看，/bin/bash，我其实一头问号。

展开 ∨

作者回复: 你指的是你们的生产环境中，容器的init进程是/bin/bash？

1



po

2020-11-24

老师我有几个疑问：

A：你说的进程被强制杀死，主要是指这个进程是init的子进程吧？如果我的应用不是多进程的应用，不会产生子进程，那就没有被强制杀死的问题了？

B：在平常写代码子进程的时候，我没有注意过写sigterm15这个情况的处理，比如Java...

展开 ∨

作者回复: > A,

是的，但进程的容器，在容器停止的时候，不会有这个问题。

> B,

如果用python启动，但是没有转发信号，那么容器结束的时候，子进程就会被强制杀死。



lihome

2020-11-23

老师能否将编译测试源码的过程放置到容器内部，避免架构差异产生的错误，比如：

运行"`docker run -d --name fwd_sig registry/fwd_sig:v1 /c-init-sig`"

出现`standard_init_linux.go:211: exec user process caused "exec format error"` 错误
原因：This can also happen when your host machine has a different architecture from your guest container image....

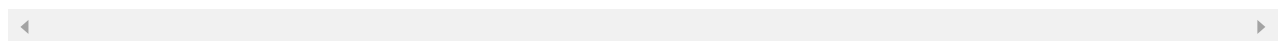
展开 ▾

作者回复: @lihome,

> 运行"`docker run -d --name fwd_sig registry/fwd_sig:v1 /c-init-sig`"

这个命令已经是在启动容器而不是在编译代码了。

请问一下，你是在Linux下运行的docker吗？



💬 1



孟令泽

2020-11-23

老师，转发sigterm信号的代码有推荐吗？

展开 ▾

作者回复: @孟令泽

可以看一下tini的代码，我在“04 | 理解进程（3）”也有提到。



💬 1



谢哈哈

2020-11-23

如果就对这个启动的程序调用kill命令，那么会输出

`ignore SIGTERM`

`catch SIGTERM`

`recieved SIGTERM`

`default SIGTERM...`

展开 ▾



💬



Geek2014

2020-11-23

输出：

`Ignore SIGTERM`

Catch SIGTERM
received SIGTERM...
展开 ▾

作者回复: Yes.



kimoti
2020-11-23

第一个是注册了Sig Ignore,所以第一个kill会被忽略。第二个是注册了自己的handler,所以会打印出receive SIGTERM,第三个是因为第二个程序里注册了default handler,所以是默认行为。

作者回复: 是的