03 | 理解进程（2）：为什么我的容器里有这么多僵尸进程?

2020-11-20 李程远

容器实战高手课 进入课程 >

讲述：李程远
时长 14:03 大小 12.87M

你好，我是程远。今天我们来聊一聊容器里僵尸进程这个问题。

说起僵尸进程，相信你并不陌生。很多面试官经常会问到这个知识点，用来考察候选人的操作系统背景。通过这个问题，可以了解候选人对 Linux 进程管理和信号处理这些基础知识的理解程度，他的基本功扎不扎实。

所以，今天我们就一起来看看容器里为什么会产生僵尸进程，然后去分析如何怎么解决。

通过这一讲，你就会对僵尸进程的产生原理有一个清晰的认识，也会更深入地理解容器 d 进程的特性。

问题再现

我们平时用容器的时候，有的同学会发现，自己的容器运行久了之后，运行 ps 命令会看到一些进程，进程名后面加了 <defunct> 标识。那么你自然会有这样的疑问，这些是什么进程呢？

你可以自己做个容器镜像来模拟一下，我们先下载这个例子，运行 `make image` 之后，再启动容器。

在容器里我们可以看到，1 号进程 fork 出 1000 个子进程。当这些子进程运行结束后，它们的进程名字后面都加了标识。

从它们的 Z stat（进程状态）中我们可以知道，这些都是僵尸进程（Zombie Process）。运行 top 命令，我们也可以看到输出的内容显示有 `1000 zombie` 进程。

复制代码

```
# docker run --name zombie-proc -d registry/zombie-proc:v1
02dec161a9e8b18922bd3599b922dbd087a2ad60c9b34afccde7c91a463bde8a
# docker exec -it zombie-proc bash
# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0   4324  1436 ?        Ss   01:23   0:00 /app-test 100
root          6  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root          7  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root          8  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root          9  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root         10  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d

…

root        999  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1000  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1001  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1002  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1003  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1004  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1005  0.0  0.0      0     0 ?        Z    01:23   0:00 [app-test] <d
root       1023  0.0  0.0  12020  3392 pts/0    Ss   01:39   0:00 bash

# top
top - 02:18:57 up 31 days, 15:17,  0 users,  load average: 0.00, 0.01, 0.00
Tasks: 1003 total,   1 running,   2 sleeping,   0 stopped, 1000 zombie
…
```

那么问题来了，什么是僵尸进程？它们是怎么产生的？僵尸进程太多会导致什么问题？想要回答这些问题，我们就要从进程状态的源头学习，看看僵尸进程到底处于进程整个生命周期里的哪一环。

知识详解

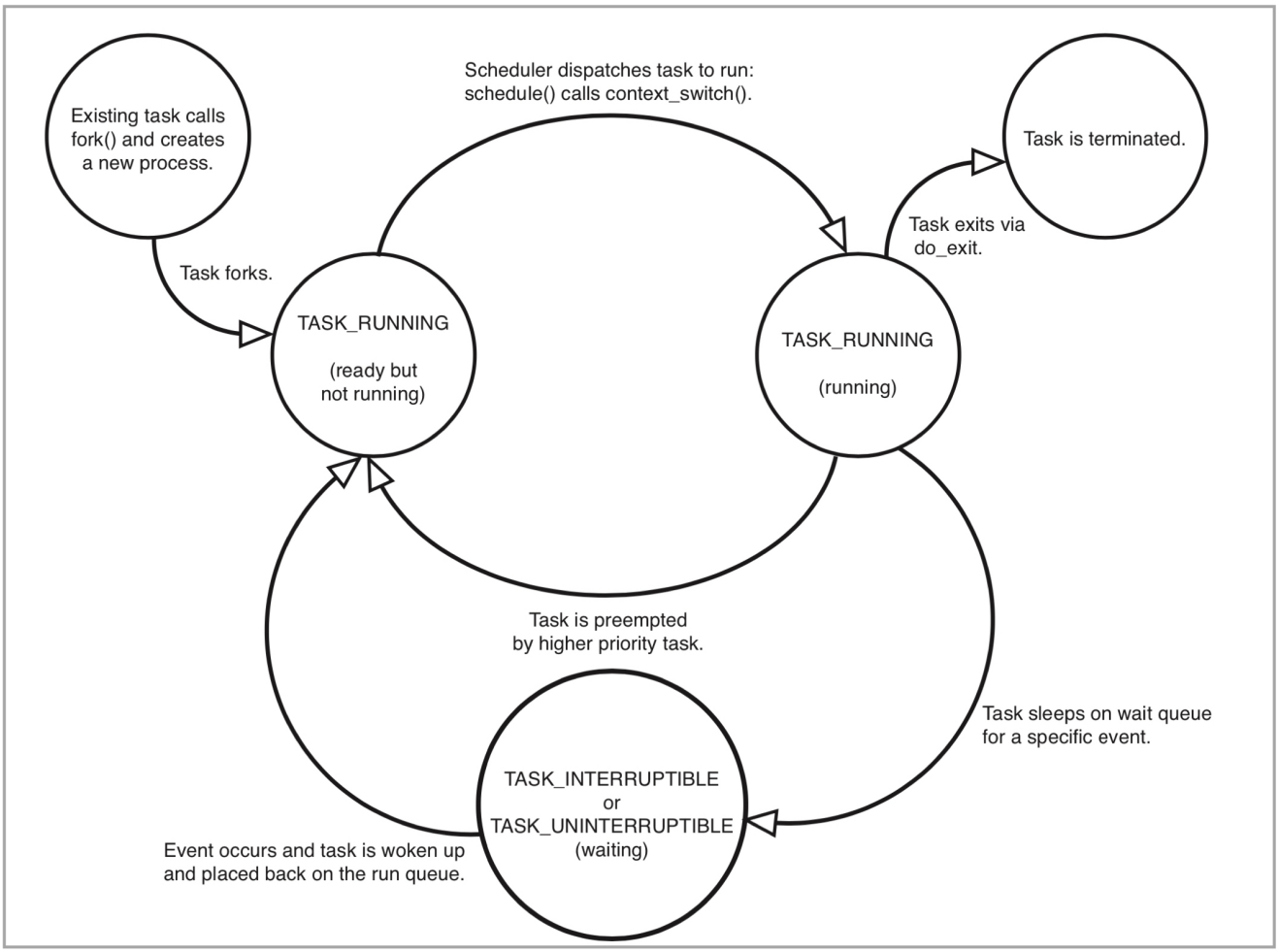
Linux 的进程状态

无论进程还是线程，在 Linux 内核里其实都是用 **task_struct{}这个结构**来表示的。它其实就是任务（task），也就是 Linux 里基本的调度单位。为了方便讲解，我们在这里暂且称它为进程。

那一个进程从创建（fork）到退出（exit），这个过程中的状态转化还是很简单的。

下面这个图是 《Linux Kernel Development》这本书里的 Linux 进程状态转化图。

我们从这张图中可以看出来，在进程“活着”的时候就只有两个状态：运行态（TASK_RUNNING）和睡眠态（TASK_INTERRUPTIBLE，TASK_UNINTERRUPTIBLE）。



那运行态和睡眠态这两种状态分别是什么意思呢？

运行态的意思是，无论进程是正在运行中（也就是获得了 CPU 资源），还是进程在 run queue 队列里随时可以运行，都处于这个状态。

我们想要查看进程是不是处于运行态，其实也很简单，比如使用 ps 命令，可以看到处于这个状态的进程显示的是 R stat。

睡眠态是指，进程需要等待某个资源而进入的状态，要等待的资源可以是一个信号量（Semaphore），或者是磁盘 I/O，这个状态的进程会被放入到 wait queue 队列里。

这个睡眠态具体还包括两个子状态：一个是可以被打断的（TASK_INTERRUPTIBLE），我们用 ps 查看到的进程，显示为 S stat。还有一个是不可被打断的（TASK_UNINTERRUPTIBLE），用 ps 查看进程，就显示为 D stat。

这两个子状态，我们在后面的课程里碰到新的问题时，会再做详细介绍，这里你只要知道这些就行了。

除了上面进程在活的时候的两个状态，进程在调用 do_exit() 退出的时候，还有两个状态。

一个是 EXIT_DEAD，也就是进程在真正结束退出的那一瞬间的状态；第二个是**EXIT_ZOMBIE 状态，这是进程在 EXIT_DEAD 前的一个状态，而我们今天讨论的僵尸进程，也就是处于这个状态中。**

限制容器中进程数目

理解了 Linux 进程状态之后，我们还需要知道，在 Linux 系统中怎么限制进程数目。因为弄清楚这个问题，我们才能更深入地去理解僵尸进程的危害。

一台 Linux 机器上的进程总数目是有限制的。如果超过这个最大值，那么系统就无法创建出新的进程了，比如你想 SSH 登录到这台机器上就不行了。

这个最大值可以我们在 /proc/sys/kernel/pid_max 这个参数中看到。

Linux 内核在初始化系统的时候，会根据机器 CPU 的数目来设置 pid_max 的值。

比如说，如果机器中 CPU 数目小于等于 32，那么 pid_max 就会被设置为 32768（32K）；如果机器中的 CPU 数目大于 32，那么 pid_max 就被设置为 N*1204（N 就是 CPU 数目）。

对于 Linux 系统而言，容器就是一组进程的集合。如果容器中的应用创建过多的进程或者出现 bug，就会产生类似 fork bomb 的行为。

这个 fork bomb 就是指在计算机中，通过不断建立新进程来消耗系统中的进程资源，它是一种黑客攻击方式。这样，容器中的进程数就会把整个节点的可用进程总数给消耗完。

这样，不但会使同一个节点上的其他容器无法工作，还会让宿主机本身也无法工作。所以对于每个容器来说，我们都需要限制它的最大进程数目，而这个功能由 pids Cgroup 这个子系统来完成。

而这个功能的实现方法是这样的：pids Cgroup 通过 Cgroup 文件系统的方式向用户提供操作接口，一般它的 Cgroup 文件系统挂载点在 /sys/fs/cgroup/pids。

在一个容器建立之后，创建容器的服务会在 /sys/fs/cgroup/pids 下建立一个子目录，就是一个控制组，控制组里**最关键的一个文件就是 pids.max**。我们可以向这个文件写入数值，而这个值就是这个容器中允许的最大进程数目。

我们对这个值做好限制，容器就不会因为创建出过多进程而影响到其他容器和宿主机了。思路讲完了，接下来我们就实际上手试一试。

下面是对一个 Docker 容器的 pids Cgroup 的操作，你可以跟着操作一下。

复制代码

```
# pwd
/sys/fs/cgroup/pids
# df ./
Filesystem     1K-blocks  Used Available Use% Mounted on
cgroup                 0     0         0    - /sys/fs/cgroup/pids
# docker ps
CONTAINER ID        IMAGE                         COMMAND                  CREATE
7ecd3aa7fdc1        registry/zombie-proc:v1   "/app-test 1000"         37 hour

# pwd
/sys/fs/cgroup/pids/system.slice/docker-7ecd3aa7fdc15a1e183813b1899d5d939beafb

# ls
cgroup.clone_children  cgroup.procs  notify_on_release  pids.current  pids.eve
# echo 1002 > pids.max
# cat pids.max
1002
```

解决问题

刚才我给你解释了两个基本概念，进程状态和进程数目限制，那我们现在就可以解决容器中的僵尸进程问题了。

在前面 Linux 进程状态的介绍里，我们知道了，僵尸进程是 Linux 进程退出状态的一种。

从内核进程的 do_exit() 函数我们也可以看到，这时候进程 task_struct 里的 mm/shm/sem/files 等文件资源都已经释放了，只留下了一个 stask_struct instance 空

壳。

就像下面这段代码显示的一样，从进程对应的 /proc/<pid> 文件目录下，我们也可以看出来，对应的资源都已经没有了。

```
# cat /proc/6/cmdline
# cat /proc/6/smaps
# cat /proc/6/maps
# ls /proc/6/fd
```

并且，这个进程也已经不响应任何的信号了，无论 SIGTERM(15) 还是 SIGKILL(9)。例如上面 pid 6 的僵尸进程，这两个信号都已经被响应了。

```
# kill -15 6
# kill -9 6
# ps -ef | grep 6
root         6     1  0 13:59 ?        00:00:00 [app-test] <defunct>
```

当多个容器运行在同一个宿主机上的时候，为了避免一个容器消耗完我们整个宿主机进程号资源，我们会配置 pids Cgroup 来限制每个容器的最大进程数目。也就是说，进程数目在每个容器中也是有限的，是一种很宝贵的资源。

既然进程号资源在宿主机上是有限的，显然残留的僵尸进程多了以后，给系统带来最大问题就是它占用了进程号。**这就意味着，残留的僵尸进程，在容器里仍然占据着进程号资源，很有可能会导致新的进程不能运转。**

这里我再次借用开头的那个例子，也就是一个产生了 1000 个僵尸进程的容器，带你理解一下这个例子中进程数的上限。我们可以看一下，1 个 init 进程 +1000 个僵尸进程 +1 个 bash 进程 ， 总共就是 1002 个进程。

如果 pids Cgroup 也限制了这个容器的最大进程号的数量，限制为 1002 的话，我们在 pids Cgroup 里可以看到，pids.current == pids.max，也就是已经达到了容器进程号数的上限。

这时候，如果我们在容器里想再启动一个进程，例如运行一下 ls 命令，就会看到 Resource temporarily unavailable 的错误消息。已经退出的无用进程，却阻碍了有用进程的启动，显然这样是不合理的。

具体代码如下：

复制代码

```
### On host
# docker ps
CONTAINER ID        IMAGE                          COMMAND             CREATED
09e6e8e16346        registry/zombie-proc:v1   "/app-test 1000"    29 minutes a

# pwd
/sys/fs/cgroup/pids/system.slice/docker-09e6e8e1634612580a03dd3496d2efed2cf2a5

# cat pids.max
1002
# cat pids.current
1002

### On Container
[root@09e6e8e16346 /]# ls
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
```

所以，接下来我们还要看看这些僵尸进程到底是怎么产生的。因为只有理解它的产生机制，我们才能想明白怎么避免僵尸进程的出现。

我们先看一下刚才模拟僵尸进程的那段小程序。这段程序里，**父进程在创建完子进程之后就不管了，这就是造成子进程变成僵尸进程的原因。**

复制代码

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>


int main(int argc, char *argv[])
{
        int i;
```

```
12          int total;
13
14          if (argc < 2) {
15                  total = 1;
16          } else {
17                  total = atoi(argv[1]);
18          }
19
20          printf("To create %d processes\n", total);
21
22          for (i = 0; i < total; i++) {
23                  pid_t pid = fork();
24
25                  if (pid == 0) {
26                          printf("Child => PPID: %d PID: %d\n", getppid(),
27                                  getpid());
28                          sleep(60);
29                          printf("Child process exits\n");
30                          exit(EXIT_SUCCESS);
31                  } else if (pid > 0) {
32                          printf("Parent created child %d\n", i);
33                  } else {
34                          printf("Unable to create child process. %d\n", i);
35                          break;
36                  }
37          }
38
39          printf("Paraent is sleeping\n");
40          while (1) {
41                  sleep(100);
42          }
43
44          return EXIT_SUCCESS;
45  }
```

前面我们通过分析，发现子进程变成僵尸进程的原因在于父进程“不负责”，那找到原因后，我们再想想，如何来解决。

其实解决思路很好理解，就好像熊孩子犯了事儿，你要去找他家长来管教，那子进程在容器里“赖着不走”，我们就需要让父进程出面处理了。

所以，在 Linux 中的进程退出之后，如果进入僵尸状态，我们就需要父进程调用 wait() 这个系统调用，去回收僵尸进程的最后的那些系统资源，比如进程号资源。

那么，我们在刚才那段代码里，主进程进入 sleep(100) 之前，加上一段 wait() 函数调用，就不会出现僵尸进程的残留了。

复制代码

```
        for (i = 0; i < total; i++) {
              int status;
              wait(&status);
        }
```

而容器中所有进程的最终父进程，就是我们所说的 init 进程，由它负责生成容器中的所有其他进程。因此，容器的 init 进程有责任回收容器中的所有僵尸进程。

前面我们知道了 wait() 系统调用可以回收僵尸进程，但是 wait() 系统调用有一个问题，需要你注意。

wait() 系统调用是一个阻塞的调用，也就是说，如果没有子进程是僵尸进程的话，这个调用就一直不会返回，那么整个进程就会被阻塞住，而不能去做别的事了。

不过这也没有关系，我们还有另一个方法处理。Linux 还提供了一个类似的系统调用 waitpid()，这个调用的参数更多。

其中就有一个参数 WNOHANG，它的含义就是，如果在调用的时候没有僵尸进程，那么函数就马上返回了，而不会像 wait() 调用那样一直等待在那里。

比如社区的一个容器 init 项目 tini。在这个例子中，它的主进程里，就是不断在调用带 WNOHANG 参数的 waitpid()，通过这个方式清理容器中所有的僵尸进程。

复制代码

```
int reap_zombies(const pid_t child_pid, int* const child_exitcode_ptr) {
        pid_t current_pid;
        int current_status;

        while (1) {
                current_pid = waitpid(-1, &current_status, WNOHANG);

                switch (current_pid) {
                        case -1:
                                if (errno == ECHILD) {
```

```
11                                      PRINT_TRACE("No child to wait");
12                                      break;
13                              }
14
15  …
```

重点总结

今天我们讨论的问题是容器中的僵尸进程。

首先，我们先用代码来模拟了这个情况，还原了在一个容器中大量的僵尸进程是如何产生的。为了理解它的产生原理和危害，我们先要掌握两个知识点：

Linux 进程状态中，僵尸进程处于 EXIT_ZOMBIE 这个状态；

容器需要对最大进程数做限制。具体方法是这样的，我们可以向 Cgroup 中 **pids.max** 这个文件写入数值（这个值就是这个容器中允许的最大进程数目）。

掌握了基本概念之后，我们找到了僵尸进程的产生原因。父进程在创建完子进程之后就不管了。

所以，我们需要父进程调用 wait() 或者 waitpid() 系统调用来避免僵尸进程产生。

关于本节内容，你只要记住下面三个主要的知识点就可以了：

1. 每一个 Linux 进程在退出的时候都会进入一个僵尸状态（EXIT_ZOMBIE）；
2. 僵尸进程如果不清理，就会消耗系统中的进程数资源，最坏的情况是导致新的进程无法启动；
3. 僵尸进程一定需要父进程调用 wait() 或者 waitpid() 系统调用来清理，这也是容器中 init 进程必须具备的一个功能。

思考题

如果容器的 init 进程创建了子进程 B，B 又创建了自己的子进程 C。如果 C 运行完之后，退出成了僵尸进程，B 进程还在运行，而容器的 init 进程还在不断地调用 waitpid()，那 C 这个僵尸进程可以被回收吗？

欢迎留言和我分享你的想法。如果你的朋友也被僵尸进程占用资源而困扰，欢迎你把这篇文章分享给他，也许就能帮他解决一个问题。

提建议

上一篇 02 | 理解进程（1）：为什么我在容器中不能kill 1号进程?

下一篇 04 | 理解进程（3）：为什么我在容器中的进程被强制杀死了?

精选留言 (24)

写留言

莫名
2020-11-20

C 应该不会被回收，waitpid 仅等待直接 children 的状态变化。

为什么先进入僵尸状态而不是直接消失？觉得是留给父进程一次机会，查看子进程的 PID、终止状态（退出码、终止原因，比如是信号终止还是正常退出等）、资源使用信息。如果子进程直接消失，那么父进程没有机会掌握子进程的具体终止情况。一般情况下，...
展开

作者回复: 谢谢 @莫名！ 很好的解释！

1 15

Helios
2020-11-20

僵尸进程也是进程，就是资源没有被回收，父进程还活着就不会被init回收。
补充一点
子进程推出的时候会给父进程发送个信号，如果父进程不处理这个信号就会变味僵尸进程。现在一般只会出现在c这种需要手动垃圾回收得语言了。
...

展开

作者回复: @Helios,
对于容器或者说pod, 我们加了pids cgroup的限制，pids.max 对于每个容器一般就是以千为单位了，这个值还是很容易达到上限的。

我们在线上看到的大量Z进程，实际的情况要复杂一些，一个进程有多个线程，主进程处于Z状态，而还有一个线程处于D状态，但是从表象查看进程状态的时候，看到都是<defunct>进程了(Z)。由于有了D的线程在里面，这时候waitpid(), 任何信号对这些进程都无效了。
这一讲，我是把Z进程的概念单独说了一下，对于D进程，它会引起其他的一些现象，我会在后面讲到。

3

水蒸蛋
2020-11-22

老师您的意思是僵尸线程默认都不会自动关闭的，全靠父进程回收，如果产生大量僵尸进程说明父进程相关回收策略有问题是吗

展开

作者回复: 对的

2

JianXu
2020-11-28

问题一：在Kubernetes 的情况下，是不是该节点上所有的容器都是kubelet 的子进程？不然kubelet 怎么来清理这些容器产生的僵尸进程呢？

问题二：在Docker 的场景下，容器第一个进程是用户自己写的进程，而该进程是不能保证在回收子进程资源上的质量的，所以才有Tinit 等工具，那为什么docker runtime 不...

展开

作者回复: > 问题一
在kuberenetes下，kubelet还是调用 containerd/runc去启动容器的，每个容器的父进程是containerd-shim, 最终shim可以回收僵尸进程。

> 问题二
docker倒是也做了这件事。 用docker启动容器的时候 加--init参数，起来的容器就强制使用tini作为init进程了。

> 问题三
Linux进程要响应SIGKILL并且执行signal handler，只有在被进程调度到的时候才可以做。对于zombie进程，它已经是不可被调度的进程了。

1

Delia
2020-11-25

我是一个Docker新手，请教一下老师，经常看到一些容器僵尸，状态栏显示：Exited (2) 10 days ago，Exited (1) 10 days ago，Exited (100) 10 days ago等等，这些容器为啥不能被回收呢？目前只能docker rm清理掉。

展开

作者回复: docker 自己没有自动清理的功能。如果是kubernetes/kubelet是会做清理。

1

上邪忘川
2020-11-22

总结一下这节课相关的东西
1.，父进程在创建完子进程之后就不管了，而每一个 Linux 进程在退出的时候都会进入一个僵尸状态，这时这些进入僵尸状态的进程就因为无法回收变成僵尸进程。
2.僵尸进程是无法直接被kill掉的，需要父进程调用wait()或watipid()回收。
3.清理僵尸进程的两个思路...

展开

作者回复: @上邪忘川, 谢谢你的总结

1

水蒸蛋
2020-11-20

老师，我还是没看明白僵尸线程的产生，它就是卡在exit zomble状态的进程，那这个产生是必然的吗，就和垃圾一样运行时间长了肯定会有一些垃圾产生，需要做的就是用主线程清理，我平时碰到这个都是重启对应的服务

展开

作者回复: 僵尸进程是进程的一个状态， 只要进程退出就会有这状态。清理的僵尸进程的是它的父进程。
这个和服务进程运行时间长了之后，通过重启进程去清理泄漏的资源，比如内存或者文件fd， 是两个概念。

1 1

kimoti
2020-11-20

想请教老师一个问题,我们现在用的容器技术比如Docker或者Kubernetes创建的容器会产生僵尸进程吗?

作者回复: @kimoti, 这个和启动容器的工具无关，而是容器中运行的程序来决定的。

1 1

1900
2020-11-20

老师你好 关于设置容器的Cgroup 中 pids.max配置， 是跟业务进程一起运行在容器中然后修改当前的容器配置，是否还有其他优雅的方式呢？

展开

作者回复: 对容器设置pids Cgroup限制，一般需要容器云平台来做，在启动容器的时候就自动的设置好，类似cpu/memory Cgroup的设置。

像Kubernetes的类似这么做，
https://github.com/kubernetes/kubernetes/commit/ecd6361ff0e8421332a50e55fcba17b823d5d338

1

夜空中最亮的星（华仔...
2020-12-01

c应该不会被回收，因为c的父亲还在，收拾c的责任在b。

一步
2020-11-29

在使用文中示例创建进程的时候，那1000个进程 会先进入 S （sleeping） 状态，等待一会后才会进入 Z （zombie） 状态？ 这是为什么的？ 这是在等待什么信号的吗？

作者回复: 进程sleep状态是这个例子代码里就是调用了sleep, 然后就退出了进程，进入zombie。

例子代码在这里：
https://github.com/chengyli/training/blob/main/init_proc/zombie_proc/app-test-nowait.c#L26

姜姜
2020-11-28

处理僵尸进程，除了显式调用wait()和waitpid()之外，还可以在父进程里显式调用signal(SIGCHLD, SIG_IGN)来忽略子进程退出的信号，这样内核将把僵尸子进程交由init进程去处理。

作者回复: 父进程只是ignore SIGCHLD, 子进程可以被init进程接管吗？

胖胖虎
2020-11-23

所以，为什么容器里面会有很多僵尸进程呢。一开始以为容器的pid 1 进程不会做进程回收导致，但是整篇文章看下来，又似乎容器的pid 1进程会waitpid。文章给的示例，是一个编程问题，但不是容器独有的，即使在容器外面跑也是会有同样的现象发生。那么本章提出的问题的答案到底是什么呢？希望老师能够帮忙解答。

展开

作者回复: 在容器里的1号进程是用户自定义的，那么容器的1号进程是否会用wait/waitpid去回收僵尸进程就不是一定的了。

争光 Alan
2020-11-23

另外感觉还是有点模糊，
僵尸进程就是子进程退出时父进程没处理导致？还是有很多情况？
另外wait()具体是操作了什么能解释一下吗？他是等待进程退出并且清理进程？

展开

作者回复: > 僵尸进程就是子进程退出时父进程没处理导致？还是有很多情况？

对于一个进程只有一个线程的时候，如果在ps里看到的是Z进程，那么它就是Z进程，就是由于父进程没有回收导致的。
不过如果一个进程中有多个线程的情况下，你在ps里看到整个进程是Z状态，但是这时候，有可能的问题是进程中有一个线程是D状态的(uninterruptible)。不过这种情况，其实是D状态的问题了。

> 他是等待进程退出并且清理进程？
对的，就是等待并清理退出的子进程。

1

争光 Alan
2020-11-23

想问下，除了父进程修复清理逻辑外，作为运维有哪些手段清理僵尸进程呢？我目前测试只找到了重启一个方法，还有别的方法吗？

展开

作者回复: 如果已经产生僵尸进程，的确没有太好的清理办法。尽量从init进程去回收僵尸进程。

1

朱雯
2020-11-22

最后我作为一个运维工程师，我还是不知道怎么处理僵死进程，第一我可能不能直接杀死他们的父进程，因为可能有用，第二，我无法kill掉他们，第三我无法修改代码，代码本身对我是黑盒子。

作者回复: 我觉得刚才 @上邪忘川的回复挺好的

3.清理僵尸进程的两个思路
（1）kill掉僵尸进程的父进程，此时僵尸进程会归附到init(1)进程下，而init进程一般都有正常的wait()或watipid()回收机制。
（2）利用dumb-init/tini之类的小型init服务来解决僵尸进程

4

朱雯
2020-11-22

关于思考题：那肯定是不行的，因为进程的父进程还在，必须好似父进程调用waitpid来操作，其他人是无权的，子进程的子进程不是我的子进程，除非父进程死去，死去后应该是由于init进程直接收养，也不会被父进程的父进程收养。

展开


朱雯
2020-11-22

老师好，
问题1:修改容器的pid.max怎么修改，我看到您直接修改宿主机的一个目录，但是我并没有类似的目录名称。
问题2: for (i = 0; i < total; i++) { int status; wait(&status); } status这个int类型的值都没有被赋值，wait又是如何知道他要回收哪个呢，同样的问题，也在于waitpid。...

展开

作者回复: @朱雯
> 问题1
你可以到 /sys/fs/cgroup/pids 目录下试试搜索一下container id, 或者带docker关键字的目录。

```
# pwd
/sys/fs/cgroup/pids
# find . -name *7bab7f79d70c*
./system.slice/docker-7bab7f79d70cbf7a59344856eecfb52c1e1d4706d3fbe3e4b74c5172ea7af541.scope
# find . -name docker*
./system.slice/docker.service
./system.slice/docker-7bab7f79d70cbf7a59344856eecfb52c1e1d4706d3fbe3e4b74c5172ea7af541.scope
./system.slice/docker.socket
```

> 问题2
wait()是不指定要回收哪个pid的，只要是它的子进程退出就可以回收。waitpid()是有参数pid的，可以指定要回收的进程。

> 问题3
把pid.maxs改成无限大，那么如果真的有海量进程在系统中，那么从内核内存，进程的调度，都会出现更多的问题或者系统直接就死了。

> 最后，这居然是一门操作系统的课程，我之前挺害怕os的，现在感觉找到一切切入点了
的确是这样的，很多容器的问题就是归结到OS的问题。学习容器也是一个很好的学习OS的切入点！

>

谢哈哈
2020-11-21

A进程也退出不了，因为B没退出，A的waitpid捕捉不到僵尸进程状态

作者回复: 嗯，
> A进程也退出不了
你这里是指C吧?

1

po
2020-11-20

老师，还有个问题，如果在发生操作系统或者容器中有很多僵尸进程，现场快速的解决办法是通过sigterm或者sigkill暂时杀掉僵尸进程解决问题吗?

展开

作者回复: SIGTERM/SIGKILL这时候是不能够杀死僵尸进程了。只能依靠父进程的wait/waitpid去回收。

2