What is a zombie (defunct) process?

SOLUTION VERIFIED - Updated January 30 2020 at 9:10 AM - English

Environment

- Red Hat Enterprise Linux

Issue

- What is a process which is represented by the " `Z` " state in a `ps` or `top` output ?
- What is the meaning of zombie process ?

Resolution

- A zombie process, also known as defunct process, is a process which has been killed or exited using the `exit` (2) system call but whose parent process has not (yet) taken notice of this through a `wait` (2) system call.
- One may deal with zombie processes in any one of the following ways:
  - Fix the parent process to make it execute `wait` (2) on child process exit
  - Kill the parent process of the zombie
  - Reboot system
  - Ignore it
- A zombie process does not consume any system resources [1] except for its entry in the process table maintained by the operating system and can be safely ignored. It will be terminated when its parent process completes execution or during system reboot.

1. This is true for a single threaded process. For multi-threaded processes, it is possible for some threads to be listed as zombies (" `Z1`) in " `ps` " output while system resources are still being used by other threads from the process. ↵

Root Cause

- A zombie process is generally created when the parent process fails to execute a wait() call on the pid of the child process when it exits (look at PPID displayed by ps -l)
- Ignoring a zombie process is safe because zombies take up little more than one extra line in the output of ps.

Diagnostic Steps

- This program creates zombie processes.

```
# cat zombie_factory.c
#include <unistd.h>

#define ZOMBIE_COUNT 100

int main(void)
{
        int i;
        for(i=0; i < ZOMBIE_COUNT; i++){
                if(fork()){
                /*parent*/
                        continue;
                }else{
                /*child*/
                        return 0;
                }
        }
        do{ sleep(1); }while(1);

  return 0;
}

# gcc -o zombie_factory zombie_factory.c

# ./zombie_factory &
[1] 9330

# ps aux | grep zombie | head
root      8812  0.0  0.5   5716  1348 pts/0    S+   21:58   0:00 vi zombie_factory.c
root      9330  0.0  0.1   1512   376 pts/2    S    22:11   0:00 ./zombie_factory
root      9331  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9332  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9333  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9334  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9335  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9336  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9337  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
root      9338  0.0  0.0      0     0 pts/2    Z    22:11   0:00 [zombie_factory] <defunct>
```

- Another program does not create zombie processes. The program is inserted wait() call:

```
# cat zombie_buster.c
#include <unistd.h>

#define ZOMBIE_COUNT 100

int main(void)
{
        int i, status, pid;
        for(i=0; i < ZOMBIE_COUNT; i++){
                if(fork()){
                /*parent*/
                        continue;
                }else{
                /*child*/
                        return 0;
                }
        }
        do{ pid = wait(&status); }while(0 < pid);

        return 0;
}

# gcc -o zombie_buster zombie_buster.c

# ./zombie_buster &
[1] 9592

# ps aux | grep zombie | head
root      9592  0.1  0.1   1512   376 pts/2    S    22:20   0:00 ./zombie_buster
root      9694  0.0  0.2   4976   756 pts/2    R+   22:20   0:00 grep zombie
```

Product(s) Red Hat Enterprise Linux Category Learn more Tags performance performance_tools process_states Tools

This solution is part of Red Hat's fast-track publication program, providing a huge library of solutions that Red Hat engineers have created while supporting our customers. To give you the knowledge you need the instant it becomes available, these articles may be presented in a raw and unedited form.

People who viewed this solution also viewed

- Why is there a watchdog zombie process in Red Hat Enterprise Linux ? — Solution - 2012年4月3日
- Jenkins is spawning many zombie processes in my OpenShift cluster — Solution - 2018年7月25日
- Scripts or programs in /etc/watchdog.d to be checked by watchdog are listed in "ps" output as "defunct" in RHEL 6 — Solution - 2015年2月6日

3 Comments

prabhat.pandey — NEWBIE — 7 Points — 10 October 2011 11:06 AM

Please include a utility in RHEL to kill the zombie processes.

Reply

Bruno Mairlot — COMMUNITY MEMBER — 42 Points — 8 October 2012 5:39 AM

You cannot kill a zombie process, because it is already dead (that's why it is called zombie). Its parent though has simply not requested to know about it.
You can only kill its parent or reboot if you want to get rid of it.

Reply

David Kalaluhi — COMMUNITY MEMBER — 43 Points — 20 September 2016 4:46 PM

This is actually incorrect. You can remove zombies more gracefully than a reboot, by forcing the parent to call wait()...actually waitpid(). If we take our zombie_factory.c above and change the count to something a little more manageable, say 2. It's much easier to deal with without batch'ing gdb commands. Once you have your zombies, you should see something like this:

```
[bash ~]$ ./zombie_factory &
[1] 23129
[bash ~]$ jobs
[1]+  Running                 ./zombie_factory &
[bash ~]$  ps aux |grep [z]ombie
dave  23129  0.0  0.0   4164   356 pts/3    S    12:30   0:00 ./zombie_factory
dave  23130  0.0  0.0      0     0 pts/3    Z    12:30   0:00 [zombie_factory] <defunct>
dave  23131  0.0  0.0      0     0 pts/3    Z    12:30   0:00 [zombie_factory] <defunct>
[bash ~]$
```

So now that we have a manageable amount of zombies to deal with, we can start killing these guys off:

```
[bash ~]$ ps -o ppid= 23131
23129
[bash ~]$ gdb
...
(gdb) attach 23129
Attaching to process 23129
...
(gdb) call waitpid(23130,0,0)
$1 = 23130
(gdb) call waitpid(23131,0,0)
$2 = 23131
(gdb) detach
Detaching from program: /home/dave/zombie_factory, process 23129
(gdb) quit
[bash ~]$ ps aux |grep zombie
dave  23141  0.0  0.0 112648   972 pts/3    S+   12:33   0:00 grep --color=auto zombie
[bash ~]$
```

Technically though, if a parent has exited, your zombies should be owned by init which should clean them up, if for some reason that's not happening, or you don't want/can't reboot. This process should work.

With the original code, that is spawning 100 Zombies, you'd definitely want to batch those pids up and have your batch file execute your call waitpid().

I hope this helps!

Also - ignoring a zombie is not really SAFE as the the Root Cause implies. While it's true it's not consuming resources, you could reach your process limit in your kernel's process table...and code throwing up tons of zombies, means there's something systemically wrong, is a bug, and needs fixed.

Reply